biomcmc-lib

0.1

Generated by Doxygen 1.8.13

Contents

1 Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

alignment_struct
Data from alignment file **6**

arg_date **8**

2 File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

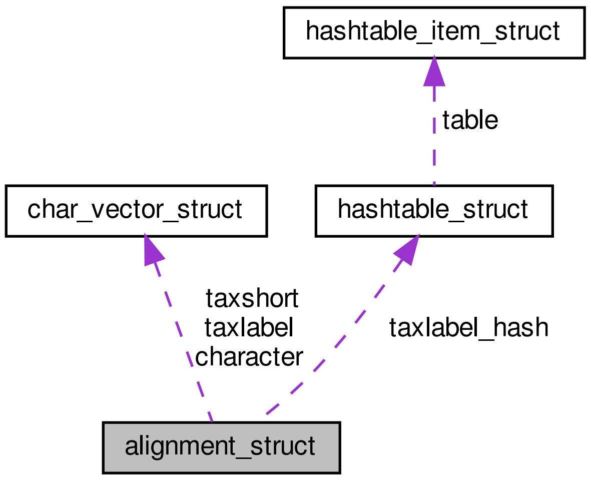
3 Data Structure Documentation

3.1 alignment_struct Struct Reference

Data from alignment file.

```
#include <alignment.h>
```

Collaboration diagram for alignment_struct:



Data Fields

- int **ntax**
- int **nchar**
- int **npat**
- char_vector character

  Number of species, sites and patterns according to sequence file.
- char_vector taxlabel

  Vector with aligned sequence for each taxon.
- char_vector taxshort

  Taxon names from file.
- hashtable taxlabel_hash

  Alias (short version) for taxon names that can be used in newick trees.
- int n_charset

  Lookup table with taxon names.
- int ∗ charset_start

  Number of gene segments (ASSUMPTIONS BLOCK).
- int ∗ **charset_end**
- bool is_aligned

  Start and end of each gene segment (from 1...NCHAR) (ASSUMPTIONS).
- int ∗ site_pattern

  FASTA files don't need to be aligned; NEXUS files do.
- int ∗ pattern_freq

  pattern, in alignment_struct::character, to which original site belongs.
- char ∗ filename

  if sequences are aligned, this is the frequency of each pattern.
- int ref_counter

  name of the original file, with extension removed

3.1.1 Detailed Description

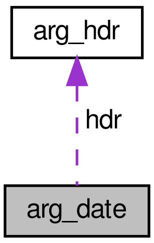
Data from alignment file.

The documentation for this struct was generated from the following file:

- alignment.h

3.2 arg_date Struct Reference

Collaboration diagram for arg_date:



Data Fields

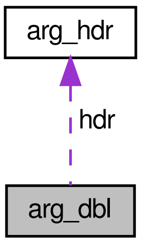
- struct arg_hdr **hdr**
- const char ∗ **format**
- int **count**
- struct tm ∗ **tmval**

The documentation for this struct was generated from the following file:

- argtable3.h

3.3 arg_dbl Struct Reference

Collaboration diagram for arg_dbl:



Data Fields

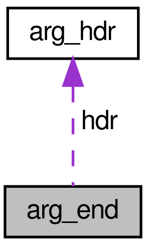
- struct arg_hdr **hdr**
- int **count**
- double ∗ **dval**

The documentation for this struct was generated from the following file:

- argtable3.h

3.4 arg_end Struct Reference

Collaboration diagram for arg_end:



Data Fields

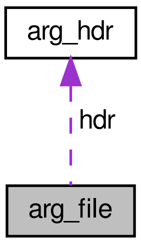
- struct arg_hdr **hdr**
- int **count**
- int ∗ **error**
- void ∗∗ **parent**
- const char ∗∗ **argval**

The documentation for this struct was generated from the following file:

- argtable3.h

3.5 arg_file Struct Reference

Collaboration diagram for arg_file:



Data Fields

- struct arg_hdr **hdr**
- int **count**
- const char ∗∗ **filename**
- const char ∗∗ **basename**
- const char ∗∗ **extension**

The documentation for this struct was generated from the following file:

- argtable3.h

3.6 arg_hdr Struct Reference

Data Fields

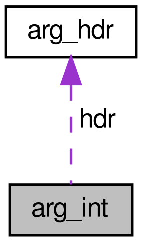
- char **flag**
- const char ∗ **shortopts**
- const char ∗ **longopts**
- const char ∗ **datatype**
- const char ∗ **glossary**
- int **mincount**
- int **maxcount**
- void ∗ **parent**
- arg_resetfn ∗ **resetfn**
- arg_scanfn ∗ **scanfn**
- arg_checkfn ∗ **checkfn**
- arg_errorfn ∗ **errorfn**
- void ∗ **priv**

The documentation for this struct was generated from the following file:

- argtable3.h

3.7 arg_int Struct Reference

Collaboration diagram for arg_int:



Data Fields

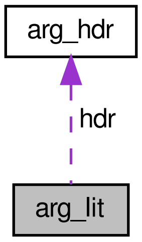
- struct arg_hdr **hdr**
- int **count**
- int ∗ **ival**

The documentation for this struct was generated from the following file:

- argtable3.h

3.8 arg_lit Struct Reference

Collaboration diagram for arg_lit:



Data Fields

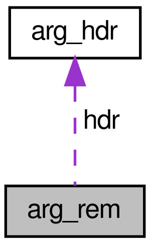
- struct arg_hdr **hdr**
- int **count**

The documentation for this struct was generated from the following file:

- argtable3.h

3.9 arg_rem Struct Reference

Collaboration diagram for arg_rem:



Data Fields

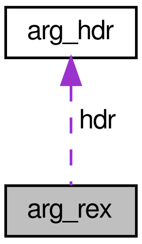
- struct arg_hdr **hdr**

The documentation for this struct was generated from the following file:

- argtable3.h

3.10 arg_rex Struct Reference

Collaboration diagram for arg_rex:



Data Fields

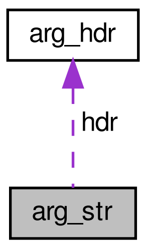
- struct arg_hdr **hdr**
- int **count**
- const char ∗∗ **sval**

The documentation for this struct was generated from the following file:

- argtable3.h

3.11 arg_str Struct Reference

Collaboration diagram for arg_str:



Data Fields

- struct arg_hdr **hdr**
- int **count**
- const char ∗∗ **sval**

The documentation for this struct was generated from the following file:

- argtable3.h

3.12 binary_parsimony_datamatrix_struct Struct Reference

used by matrix representation with parsimony (01 10 11 sequences)

```
#include <parsimony.h>
```

Data Fields

- int **ntax**
- int **nchar**
- int i

  number of taxa, distinct sites (patterns), and index to current (last) column
- bool ∗∗ s

  1 (01) and 2 (10) are the two binary states, with 3 (11) being undetermined
- int ∗ **freq**
- int freq_sum

  frequency of pattern.
- int ∗ occupancy

  how many species represented by each bipartition
- uint32_t ∗ col_hash

  hash value of each column, to speed up comparisons
- int ref_counter

  how many places have a pointer to this instance

3.12.1 Detailed Description

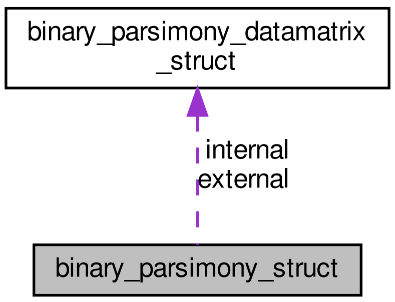
used by matrix representation with parsimony (01 10 11 sequences)

The documentation for this struct was generated from the following file:

- parsimony.h

3.13 binary_parsimony_struct Struct Reference

Collaboration diagram for binary_parsimony_struct:



Data Fields

- int ∗ score

  parsimony score per pattern
- binary_parsimony_datamatrix **external**
- binary_parsimony_datamatrix internal

  binary matrices for leaves and for internal nodes
- double **costs** [4]
- int ref_counter

  how many places have a pointer to this instance

The documentation for this struct was generated from the following file:

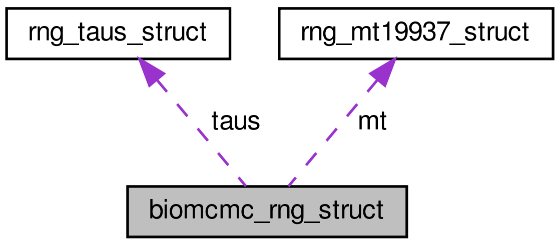
- parsimony.h

3.14 biomcmc_rng_struct Struct Reference

Random number structure (combined Tausworthe algorithm)

```
#include <random_number.h>
```

Collaboration diagram for biomcmc_rng_struct:



Data Fields

- rng_taus_struct **taus**
- rng_mt19937_struct mt

  Tausworthe linear feedback shift-register from GSL.
- uint64_t bit32

  64 bits Mersenne Twister from Matsumoto's webpage
- bool have_bit32

  temporary values when only 32 bits are necessary
- double rnorm32

  when using 32 bits we first check if we have one stored
- double **rnorm64**
- bool have_rnorm32

  stored standard normal random values with 32 and 52 bits of precision
- bool **have_rnorm64**

3.14.1 Detailed Description

Random number structure (combined Tausworthe algorithm)

The documentation for this struct was generated from the following file:

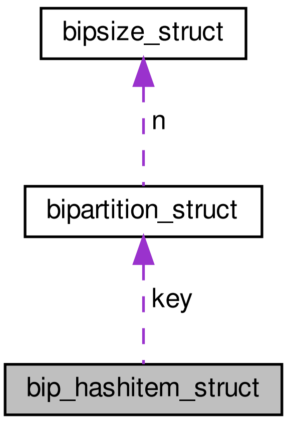
- random_number.h

3.15 bip_hashitem_struct Struct Reference

key (bipartition) and value (frequency) pair for hash table of bipartitions

```
#include <hashtable.h>
```

Collaboration diagram for bip_hashitem_struct:



Data Fields

- bipartition **key**
- int count

    pointer to bipartition (must update ref_counter)

3.15.1 Detailed Description

key (bipartition) and value (frequency) pair for hash table of bipartitions

The documentation for this struct was generated from the following file:

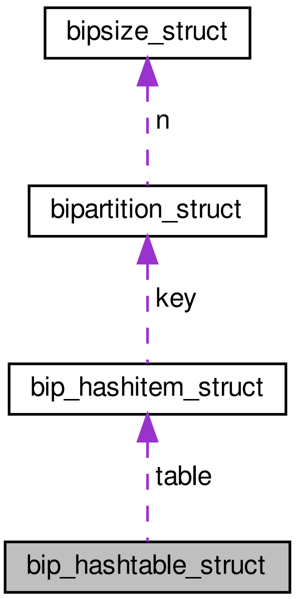
- hashtable.h

3.16 bip_hashtable_struct Struct Reference

Hash table of bipartitions (see hashtable.h for original version, with string keys and integer values)

```
#include <hashtable.h>
```

Collaboration diagram for bip_hashtable_struct:



Data Fields

- int **size**
- int probelength

  Table size.
- int maxfreq

  Number of collisions before empty slot is found.
- uint32_t h

  frequency (integer) of most frequent bipartition
- uint32_t a1

  Value set by hash(). Used in hash1() and hash2() to avoid calling hash() again.
- uint32_t **a2**
- uint32_t **b1**
- uint32_t **b2**
- uint32_t P

  Random values used in hash functions.
- bip_hashitem ∗ **table**
- int ref_counter

  Vector with key/value pairs.

3.16.1 Detailed Description

Hash table of bipartitions (see hashtable.h for original version, with string keys and integer values)

The documentation for this struct was generated from the following file:

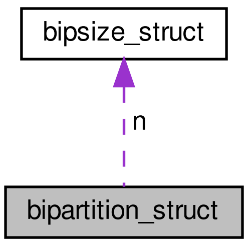
- hashtable.h

3.17 bipartition_struct Struct Reference

Bit-string representation of splits.

```
#include <bipartition.h>
```

Collaboration diagram for bipartition_struct:



Data Fields

- uint64_t $*$ bs

  Representation of a bipartition by a vector of integers (bitstrings).
- int n_ones

  Counter (number of "one"s)
- bipsize n

  number of bits (leaves), vector size and mask
- int ref_counter

  How many times this struct is being referenced.

3.17.1 Detailed Description

Bit-string representation of splits.

The documentation for this struct was generated from the following file:

- bipartition.h

3.18 bipsize_struct Struct Reference

Data Fields

- uint64_t mask

    mask to make sure we consider only active positions (of last bitstring)
- int ints

    Vector size and total number of elements (n_ints = n_bits/(8∗sizeof(long long)) +1).
- int **bits**
- int **original_size**
- int ref_counter

    How many times this struct is being referenced.

The documentation for this struct was generated from the following file:

- bipartition.h

3.19 char_vector_struct Struct Reference

vector of strings (char vectors) of variable length

```
#include <char_vector.h>
```

Data Fields

- char ∗∗ **string**
- int nstrings

    vector of strings
- size_t ∗ alloc

    how many strings
- size_t ∗ nchars

    in some cases (e.g. huge fasta files) we need to reduce calls to realloc()
- int ref_counter

    length of allocated memory for each string excluding the ending '\0' (the actual size in use needs strlen() or a call to char_vector_compress() over the structure)
- int next_avail

    how many times this char_vector_struct is being used

3.19.1 Detailed Description

vector of strings (char vectors) of variable length

The documentation for this struct was generated from the following file:

- char_vector.h

3.20 charvec_str Struct Reference

Data Fields

- char ∗ **s**
- int **idx**
- size_t **nchars**

The documentation for this struct was generated from the following file:

- char_vector.c

3.21 discrete_sample_struct Struct Reference

Data Fields

- size_t **K**
- size_t ∗ **A**
- double ∗ **F**

The documentation for this struct was generated from the following file:

- prob_distribution.h

3.22 distance_generator_struct Struct Reference

Data Fields

- int **n_samples**
- int **n_distances**
- int **which_distance**
- double ∗∗ **dist**
- bool ∗ **cached**
- void ∗ **data**
- void(∗ **distance_function**)(void ∗, int, int, double ∗)
- int **ref_counter**

The documentation for this struct was generated from the following file:

- distance_generator.h

3.23 distance_matrix_struct Struct Reference

Data Fields

- int **size**
- double $**$ d

  number of sequences to calculate distances
- double mean_K2P_dist

  pairwise distance matrix (upper) and ti/tv rate ratio (lower triangle) for K2P formula for alignments
- double var_K2P_dist

  average pairwise distance from K2P model
- double mean_JC_dist

  variance in pairwise distance from K2P model
- double mean_R

  average pairwise distance from JC model
- double var_R

  average K2P transition/transversion ratio from pairwise distances
- double freq [20]

  variance in K2P transition/transversion ratio from pairwise distances
- double $*$ fromroot

  empirical equilibrium frequencies
- int $*$ idx

  distance from root (used to calculate distance between tree leaves)
- int $*$ **i_l**
- int $*$ **i_r**
- int ref_counter

  aux vectors for finding leaves spanned by subtrees on any node

The documentation for this struct was generated from the following file:

- distance_matrix.h

3.24 dsample_stack_struct Struct Reference

Data Fields

- size_t **N**
- size_t $*$ **v**
- size_t **i**

The documentation for this struct was generated from the following file:

- prob_distribution.c

3.25 edgearray_item Struct Reference

Data Fields

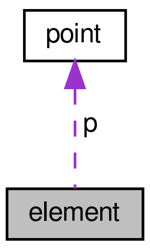
- int **id**
- double **distance**

The documentation for this struct was generated from the following file:

- clustering_goptics.c

3.26 element Struct Reference

Collaboration diagram for element:



Data Fields

- point ∗ **p**

The documentation for this struct was generated from the following file:

- clustering_goptics.c

3.27 empfreq_double_element Struct Reference

Data Fields

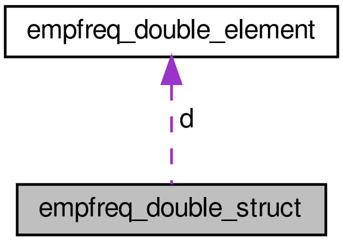
- double **freq**
- int **idx**

The documentation for this struct was generated from the following file:

- empirical_frequency.h

3.28 empfreq_double_struct Struct Reference

Collaboration diagram for empfreq_double_struct:



Data Fields

- empfreq_double_element ∗ **d**
- int **n**
- double **min**
- double max

    Min value for index.
- int ref_counter

    Max value for index.

The documentation for this struct was generated from the following file:

- empirical_frequency.h

3.29 empfreq_element Struct Reference

Data Fields

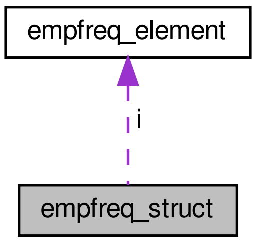
- int **freq**
- int **idx**

The documentation for this struct was generated from the following file:

- empirical_frequency.h

3.30 empfreq_struct Struct Reference

Collaboration diagram for empfreq_struct:



Data Fields

- empfreq_element ∗ **i**
- int **n**
- int **min**
- int max

  Min value for index.
- int ref_counter

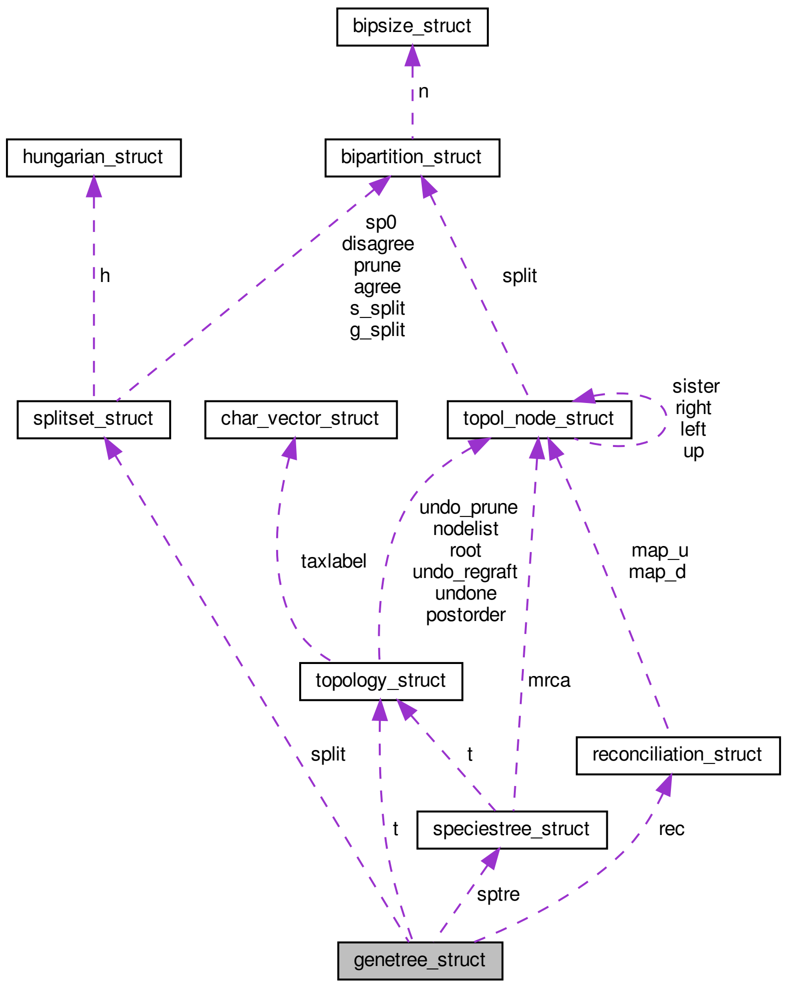
  Max value for index.

The documentation for this struct was generated from the following file:

- empirical_frequency.h

3.31 genetree_struct Struct Reference

Collaboration diagram for genetree_struct:



Data Fields

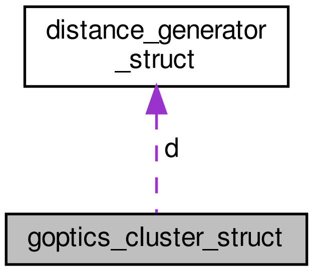
- topology **t**
- reconciliation **rec**
- speciestree **sptre**
- splitset **split**
- int ∗ **distance**
- int ∗ **minmax**
- int **ref_counter**

The documentation for this struct was generated from the following file:

- genetree.h

3.32 goptics_cluster_struct Struct Reference

Collaboration diagram for goptics_cluster_struct:



Data Fields

- int ∗ **Va_i**
- int ∗ **Va_n**
- double **epsilon**
- int **min_points**
- int **num_edges**
- int **n_clusters**
- int ∗ **order**
- int **n_order**
- int ∗ **cluster**
- double ∗ **core_distance**
- double ∗ **reach_distance**
- double **max_distance**
- bool ∗ **core**
- void ∗ **Ea**
- void ∗ **heap**
- void ∗ **points**
- double **timing_secs**
- distance_generator **d**

The documentation for this struct was generated from the following file:

- clustering_goptics.h

3.33 hashtable_item_struct Struct Reference

key/value pair for hash table

```
#include <hashtable.h>
```

Data Fields

- char ∗ key

    String (vector of char).
- int value

    Integer (position in vector where hashtable_item_struct::key can be found)

3.33.1 Detailed Description

key/value pair for hash table

The documentation for this struct was generated from the following file:

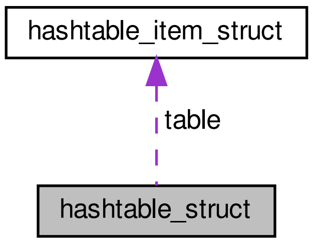
- hashtable.h

3.34 hashtable_struct Struct Reference

Hash table (vector indexed by strings).

```
#include <hashtable.h>
```

Collaboration diagram for hashtable_struct:



Data Fields

- int size

    Table size.
- int probelength

    Number of collisions before empty slot is found.
- uint32_t h

    Value set by hash(). Used in hash1() and hash2() to avoid calling hash() again.
- uint32_t a1

    Random values used in hash functions.
- uint32_t a2

    Random values used in hash functions.

- uint32_t b1

  Random values used in hash functions.
- uint32_t b2

  Random values used in hash functions.
- uint32_t P

  Random values used in hash functions.
- hashtable_item ∗ table

  Vector with key/value pairs.
- int ref_counter

  Counter of how many external references (structures sharing this hashtable) to avoid deletion.

3.34.1 Detailed Description

Hash table (vector indexed by strings).

The documentation for this struct was generated from the following file:

- hashtable.h

3.35 hll_estimate_s Struct Reference

```
#include <hll.h>
```

Data Fields

- double **alpha**
- uint16_t n_buckets
- uint16_t n_empty_buckets
- uint64_t estimate
- uint64_t hll_estimate
- uint64_t small_range_estimate
- uint64_t large_range_estimate

3.35.1 Detailed Description

Estimation result data structure

3.35.2 Field Documentation

3.35.2.1 n_buckets

```
uint16_t hll_estimate_s::n_buckets
```

Alpha

3.35.2.2 n_empty_buckets

```
uint16_t hll_estimate_s::n_empty_buckets
```

Number of buckets

3.35.2.3 estimate

```
uint64_t hll_estimate_s::estimate
```

Number of empty buckets

3.35.2.4 hll_estimate

```
uint64_t hll_estimate_s::hll_estimate
```

Final estimated cardinality

3.35.2.5 small_range_estimate

```
uint64_t hll_estimate_s::small_range_estimate
```

HLL estimated cardinaloty, before any correction

3.35.2.6 large_range_estimate

```
uint64_t hll_estimate_s::large_range_estimate
```

Small range estimated cardinality

The documentation for this struct was generated from the following file:

- hll.h

3.36 hll_s Struct Reference

Data Fields

- double **alpha**
- size_t **n_buckets**
- uint8_t * **buckets**
- hll_hash_function_t **hash_function**

The documentation for this struct was generated from the following file:

- hll.c

3.37 hungarian_struct Struct Reference

Data Fields

- int ∗∗ **cost**
- int size

  cost matrix
- int initial_cost

  assignment size. Cost is a square matrix, so size should be an overestimate where "missing" nodes are added w/ cost zero
- int final_cost

  sum of lowest input cost values for each column. The hungarian method rescales them so that minimum per column is zero
- int ∗ col_mate

  our final cost is on rescaled cost matrix, therefore to restore the "classical" optimal cost one should sum it with initial_cost
- int ∗ **unchosen_row**
- int ∗ **slack_row**
- int ∗ **row_mate**
- int ∗ **parent_row**
- double ∗∗ dcost

  col_mate[row] with column match for row
- double **initial_dcost**
- double **final_dcost**
- double ∗ row_dec_d

  costs when working with float numbers instead of integers
- double ∗ **col_inc_d**
- double ∗ **slack_d**
- int ∗ **row_dec**
- int ∗ **col_inc**
- int ∗ **slack**
- bool **is_double**

The documentation for this struct was generated from the following file:

- lowlevel.h

3.38 kmer_params_struct Struct Reference

Data Fields

- uint64_t **mask1** [7]
- uint64_t **mask2** [7]
- uint8_t **n1**
- uint8_t **n2**
- uint8_t **shift1** [7]
- uint8_t **shift2** [7]
- uint8_t **size** [14]
- uint8_t **nbytes** [14]
- uint32_t **seed** [14]
- uint64_t(∗ **hashfunction**)(const void ∗, const size_t, const uint32_t)
- int **dense**
- int kmer_class_mode

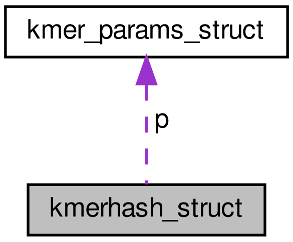
  4bits per base or 2bits or 1 bit (GC content)
- int **ref_counter**

The documentation for this struct was generated from the following file:

- kmerhash.h

3.39 kmerhash_struct Struct Reference

Collaboration diagram for kmerhash_struct:



Data Fields

- kmer_params **p**
- uint64_t ∗ **forward**
- uint64_t ∗ **reverse**
- uint64_t ∗ **hash**
- uint64_t ∗ **kmer**
- int n_hash

    hash = 4mer, 8mer, etc. hashed ; kmer = original bitstring OR its complement, masked
- int **n_f**
- char ∗ dna

    n_f = 2 (128bits)
- size_t **i**
- size_t **n_dna**
- int **ref_counter**

The documentation for this struct was generated from the following file:

- kmerhash.h

3.40 longoptions Struct Reference

Data Fields

- int **getoptval**
- int **noptions**
- struct option ∗ **options**

The documentation for this struct was generated from the following file:

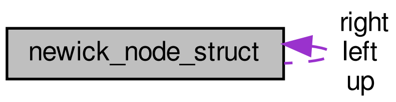
- argtable3.c

3.41 newick_node_struct Struct Reference

newick trees have minimal information, unlike topology_struct

```
#include <read_newick_trees.h>
```

Collaboration diagram for newick_node_struct:



Data Fields

- newick_node **up**
- newick_node **right**
- newick_node **left**
- int id

  Parent and children nodes.
- double branch_length

  Initial pre-order numbering of node.
- char ∗ taxlabel

  Branch length from node to node->up.

3.41.1 Detailed Description

newick trees have minimal information, unlike topology_struct

The documentation for this struct was generated from the following file:

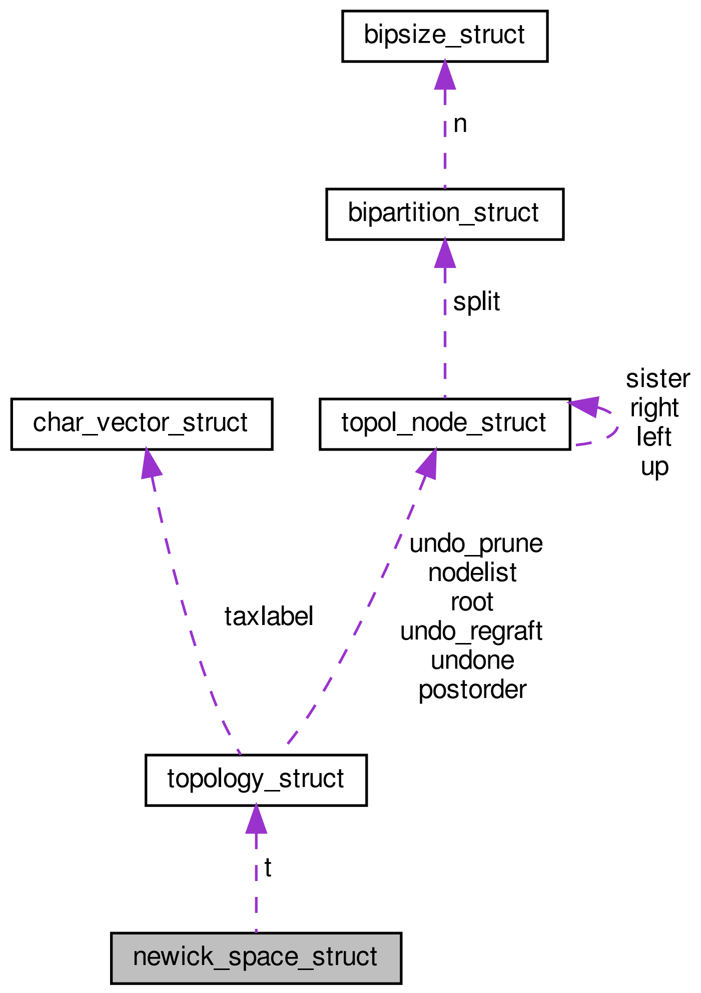
- read_newick_trees.h

3.42 newick_space_struct Struct Reference

Collection of topologies from tree file. Each topology will have its own char_vector.

```
#include <newick_space.h>
```

Collaboration diagram for newick_space_struct:



Data Fields

- int **ntrees**
- topology ∗ t

  Number of trees originally in nexus file and compacted (only distinct topologies).
- int ref_counter

  Vector of trees originally in nexus file and compacted.

3.42.1 Detailed Description

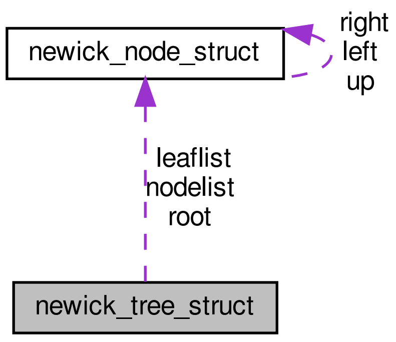
Collection of topologies from tree file. Each topology will have its own char_vector.

The documentation for this struct was generated from the following file:

- newick_space.h

3.43 newick_tree_struct Struct Reference

Collaboration diagram for newick_tree_struct:



Data Fields

- newick_node ∗ **nodelist**
- newick_node ∗ leaflist

  Vector with pointers to every internal node.
- newick_node root

  Vector with pointers to tree leaves.
- int nnodes

  Pointer to root node.
- int **nleaves**

The documentation for this struct was generated from the following file:

- read_newick_trees.h

3.44 point Struct Reference

Data Fields

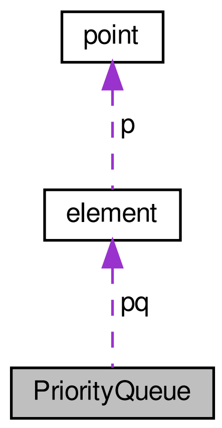
- int **id**
- double **coreDist**
- double **reachDist**
- bool **processed**
- int **pqPos**

The documentation for this struct was generated from the following file:

- clustering_goptics.c

3.45 PriorityQueue Struct Reference

Collaboration diagram for PriorityQueue:



Data Fields

- element ∗ **pq**
- int **n**
- int **heap_size**

The documentation for this struct was generated from the following file:

- clustering_goptics.c

3.46 privhdr Struct Reference

Data Fields

- const char ∗ **pattern**
- int **flags**

The documentation for this struct was generated from the following file:

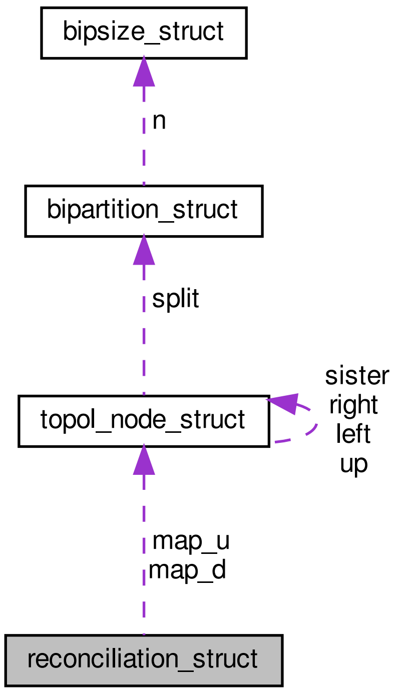
- argtable3.c

3.47 reconciliation_struct Struct Reference

mapping between gene tree nodes (this) and (external) species tree nodes

```
#include <genetree.h>
```

Collaboration diagram for reconciliation_struct:



Data Fields

- topol_node ∗ **map_d**
- topol_node ∗ map_u

  Mapping of all nodes from gene to species (the first gene::nnodes are fixed)
- int ∗ sp_id

  Mapping of all nodes from gene to species, assuming gene tree is upside down (unrooted TESTING version)
- int ∗ sp_count

  mapping of gene (this tolopogy) leaf to ID of taxon in species tree
- int sp_size

  how many copies of each species are present in this gene (used by deepcoal)
- int size_diff

  effective number of species present in gene family
- int ∗ dup

  twice the difference in number of leaves between gene tree and (reduced/effective) species tree
- int ∗ ndup_d

indexes of duplication nodes on gene tree, and number of such nodes (unused for now)

- int ∗ ndup_u

  number of duplications below node (edge above node, since struct assume node == edge above it)
- int ∗ nlos_d

  number of duplications above node (edge upside down, thus "children" are 'up' and 'sister')
- int ∗ nlos_u

  number of losses below node and edge above node
- int ndups

  number of losses above node, including edge above it
- int nloss

  minimum number of duplications over all possible rootings, acc. to reconciliation_struct::dup
- int ndcos

  number of losses corresponding to rooting (edge) that minimizes duplications

3.47.1 Detailed Description

mapping between gene tree nodes (this) and (external) species tree nodes

The documentation for this struct was generated from the following file:

- genetree.h

3.48 rng_diaconis_struct Struct Reference

Persi Diaconis' lagged Fibonacci.

```
#include <random_number_gen.h>
```

Data Fields

- uint32_t **x** [128]
- int **n**

3.48.1 Detailed Description

Persi Diaconis' lagged Fibonacci.

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.49 rng_gfsr4_struct Struct Reference

GFSR4 implementation from GSL.

```
#include <random_number_gen.h>
```

Data Fields

- uint32_t **x** [16384]
- int **n**

3.49.1 Detailed Description

GFSR4 implementation from GSL.

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.50 rng_lfib4_struct Struct Reference

Marsaglia's LFIB4 lagged Fibonacci using addition.

```
#include <random_number_gen.h>
```

Data Fields

- uint32_t **x** [256]
- int **n**

3.50.1 Detailed Description

Marsaglia's LFIB4 lagged Fibonacci using addition.

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.51 rng_mt19937_struct Struct Reference

MT19937-64, the Mersenne Twister for 64 bits.

```
#include <random_number_gen.h>
```

Data Fields

- uint64_t **x** [312]
- int **n**

3.51.1 Detailed Description

MT19937-64, the Mersenne Twister for 64 bits.

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.52 rng_mt19937ar_struct Struct Reference

MT19937, the original Mersenne Twister (for 32 bits); the name "ar" comes from "array".

```
#include <random_number_gen.h>
```

Data Fields

- uint32_t **x** [624]
- int **n**

3.52.1 Detailed Description

MT19937, the original Mersenne Twister (for 32 bits); the name "ar" comes from "array".

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.53 rng_swb_struct Struct Reference

Marsaglia's Subtract-with-borrow generator.

```
#include <random_number_gen.h>
```

Data Fields

- uint32_t **x** [258]
- int **n**

3.53.1 Detailed Description

Marsaglia's Subtract-with-borrow generator.

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.54 rng_taus_struct Struct Reference

Data Fields

- uint64_t **x** [30]
- int **n**

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.55 rng_tt800_struct Struct Reference

tt800 (small cousin of MT19937, the Mersenne Twister)

```
#include <random_number_gen.h>
```

Data Fields

- uint32_t **x** [25]
- int **n**

3.55.1 Detailed Description

tt800 (small cousin of MT19937, the Mersenne Twister)

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.56 rng_well1024_struct Struct Reference

Data Fields

- uint32_t **x** [32]
- int **n**

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.57 rng_xorshift_struct Struct Reference

Data Fields

- uint64_t **x** [65]
- int **n**

The documentation for this struct was generated from the following file:

- random_number_gen.h

3.58 spdist_matrix_struct Struct Reference

Data Fields

- int **size**
- int **n_missing**
- double ∗ **mean**
- double ∗ **min**
- int ∗ count

  mean or min distances across possibilities (within loci)
- bool ∗ species_present

  how many times this pairwise comparison appears (between or within loci)
- int ref_counter

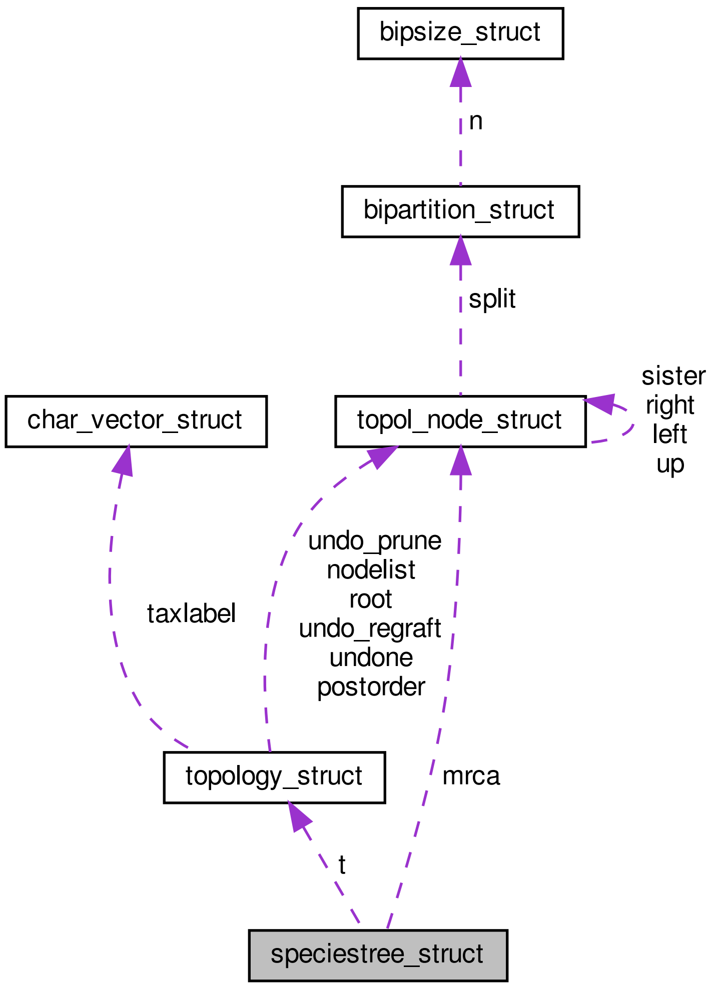
  boolean marking if species is present at all in this matrix

The documentation for this struct was generated from the following file:

- distance_matrix.h

3.59 speciestree_struct Struct Reference

Collaboration diagram for speciestree_struct:



Data Fields

- topology **t**
- topol_node ∗ **mrca**
- int ∗ spnames_order

  triangular matrix of topol_nodes (LCA between topol_node::id (i-1) and j) in one dimension
- int ref_counter

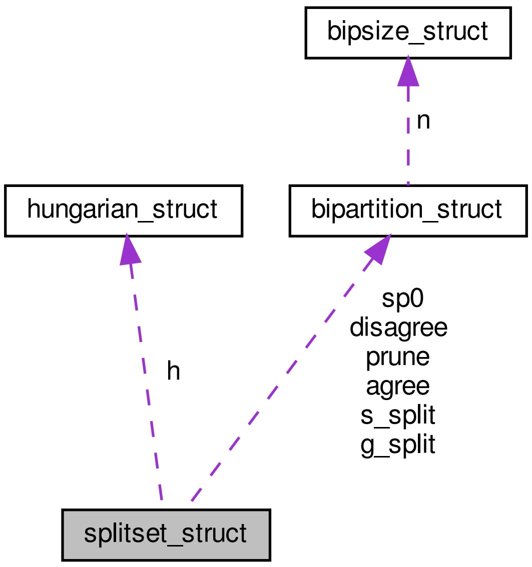
  Length+lexico order of sptree leaf names (not used unless added by user, when arbitrary leaf ordering is requested)

The documentation for this struct was generated from the following file:

- genetree.h

3.60 splitset_struct Struct Reference

Collaboration diagram for splitset_struct:



Data Fields

- int **size**
- int **spsize**
- int **spr**
- int **spr_extra**
- int **rf**
- int **hdist**
- int **hdist_reduced**
- int n_g

    spr, extra prunes for spr, rf distances and hdist=assignment cost
- int **n_s**
- int **n_agree**
- int **n_disagree**
- bipartition ∗ **g_split**
- bipartition ∗ **s_split**
- bipartition ∗ **agree**
- bipartition ∗ **disagree**
- bipartition ∗ **sp0**
- bipartition **prune**
- hungarian **h**
- bool **match**

The documentation for this struct was generated from the following file:

- genetree.h

3.61 tagTRexNode Struct Reference

Data Fields

- TRexNodeType **type**
- int **left**
- int **right**
- int **next**

The documentation for this struct was generated from the following file:

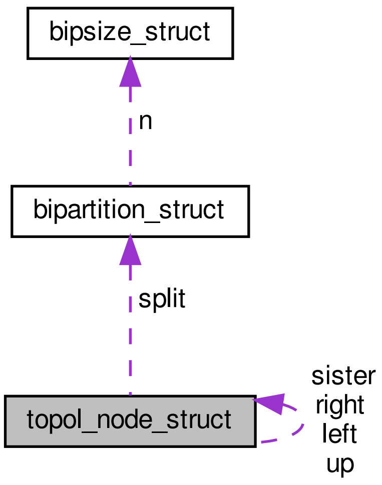
- argtable3.c

3.62 topol_node_struct Struct Reference

Information of a node (binary tree).

```
#include <topology_common.h>
```

Collaboration diagram for topol_node_struct:



Data Fields

- topol_node **up**
- topol_node **right**
- topol_node **left**
- topol_node **sister**
- int id

  Parent, children and sister nodes.

- int **level**
- int mid [5]

  Node ID (values smaller than nleaves indicate leaves) and distance from root.
- bool internal

  Mapping between nodes and postorder vectors [0,1] (postorder, undone); idx for deep coal [2,3] and losses [4].
- bool u_done

  If internal node, TRUE; if leaf, FALSE.
- bool d_done

  Has the topology up this edge (eq. to node) changed? (needed in likelihood calc)
- bipartition split

  Has the topology down this edge (eq. to node) changed? (needed in likelihood calc)

3.62.1 Detailed Description

Information of a node (binary tree).

The documentation for this struct was generated from the following file:

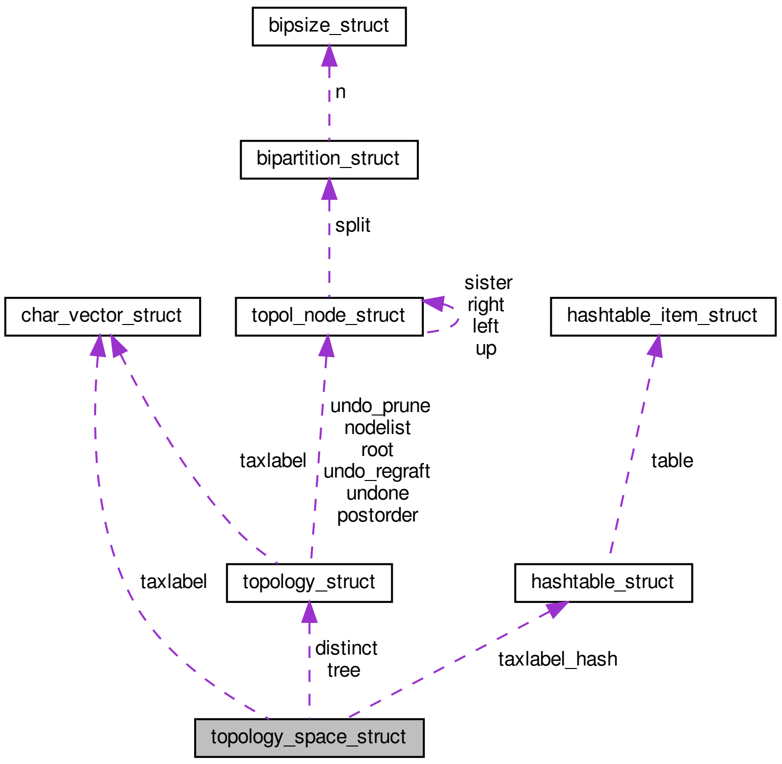
- topology_common.h

3.63 topology_space_struct Struct Reference

Collection of topologies from tree file. When topologies have no branch lengths we store only unique topologies.

```
#include <topology_space.h>
```

Collaboration diagram for topology_space_struct:



Data Fields

- int **ntrees**
- int **ndistinct**
- topology ∗ tree

  Number of trees originally in nexus file and compacted (only distinct topologies).
- topology ∗ **distinct**
- double ∗ freq

  Vector of trees originally in nexus file and compacted.
- char_vector taxlabel

  frequency of each distinct topology (add up to one)
- hashtable taxlabel_hash

  Taxon names.
- bool is_rooted

  Lookup table with taxon names.
- char ∗ filename

  If trees are unrooted, then branch lengths must be accounted for in some comparisons.

3.63.1 Detailed Description

Collection of topologies from tree file. When topologies have no branch lengths we store only unique topologies.

The documentation for this struct was generated from the following file:

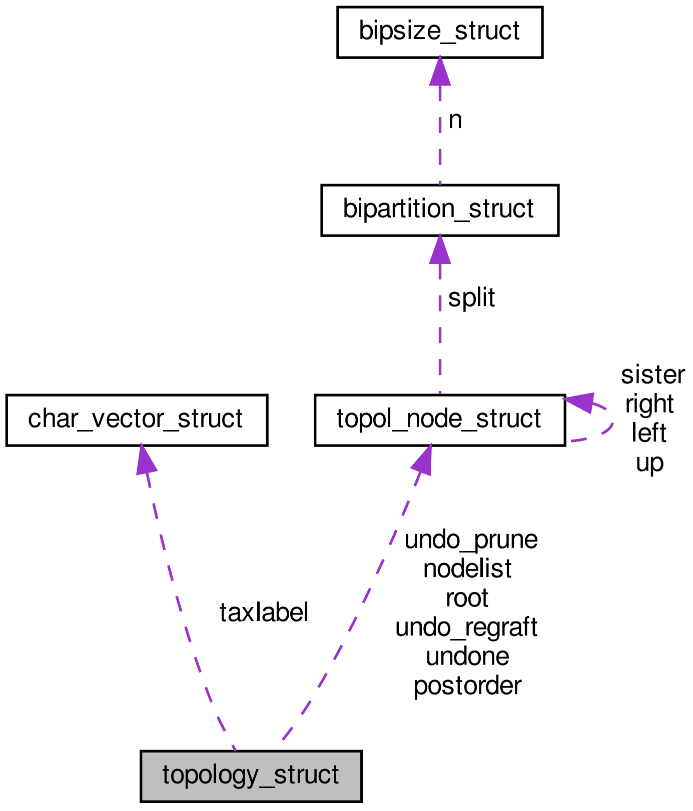
- topology_space.h

3.64 topology_struct Struct Reference

Binary unrooted topology (rooted at leaf with ID zero)

```
#include <topology_common.h>
```

Collaboration diagram for topology_struct:



Data Fields

- topol_node $*$ **nodelist**
- double $*$ blength

  vector of nodes (the first L are the leaves).
- int id

  Branch lengths, with mean, min, max vectors for topology_space.
- topol_node root

  topology ID (should be updated by hand, e.g. by functions in topology_space.c)
- int nleaves

  Pointer to root node.
- int nnodes

  Number of leaves L.
- topol_node undo_prune

  Number of nodes, including leaves ($2L-1$ for a binary rooted tree).
- topol_node undo_regraft

  How to revert most recent SPR move (prune node).
- bool undo_lca

  How to revert most recent SPR move (regraft node).

- topol_node ∗ postorder

    revert SPR move is lca type or not
- topol_node ∗ undone

    pointers to all internal nodes in postorder (from last to first is preorder)
- int n_undone

    pointers to outdated nodes in postorder (from last to first is preorder)
- uint32_t hashID1

    number of outdated nodes (which need likelihood calc etc) in topology_struct::undone.
- uint32_t **hashID2**
- bool traversal_updated

    hash values of tree, ideally a unique value for each tree (collisions happen...)
- int ref_counter

    zero if postorder[] vector needs update, one if we can use postdorder[] to traverse tree
- char_vector taxlabel

    number of references of topology (how many places are pointing to it)
- int ∗ index

    Taxon names (just a pointer; actual values are setup by newick_tree_struct or alignment_struct)
- bool quasirandom

    sandbox vector used in spr moves / quasirandom tree shuffle just to avoid recurrent allocation

3.64.1 Detailed Description

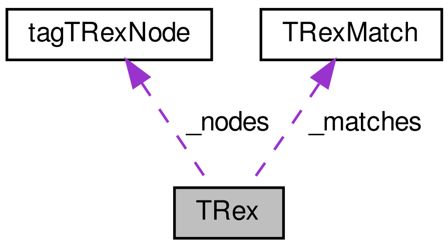
Binary unrooted topology (rooted at leaf with ID zero)

The documentation for this struct was generated from the following file:

- topology_common.h

3.65 TRex Struct Reference

Collaboration diagram for TRex:



Data Fields

- const TRexChar ∗ **_eol**
- const TRexChar ∗ **_bol**
- const TRexChar ∗ **_p**
- int **_first**
- int **_op**
- TRexNode ∗ **_nodes**
- int **_nallocated**
- int **_nsize**
- int **_nsubexpr**
- TRexMatch ∗ **_matches**
- int **_currsubexp**
- void ∗ **_jmpbuf**
- const TRexChar ∗∗ **_error**
- int **_flags**

The documentation for this struct was generated from the following file:

- argtable3.c

3.66 TRexMatch Struct Reference

Data Fields

- const TRexChar ∗ **begin**
- int **len**

The documentation for this struct was generated from the following file:

- argtable3.c

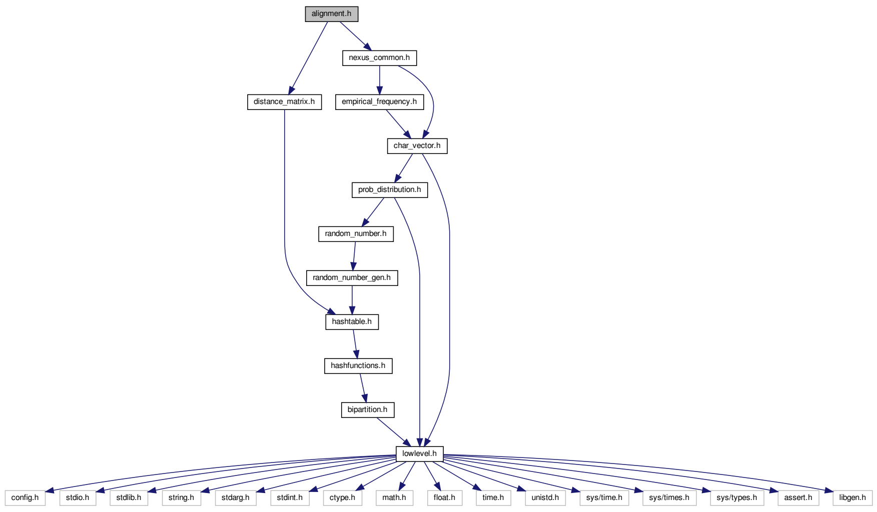
4 File Documentation

4.1 alignment.h File Reference

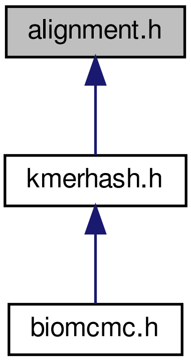
File handling functions and calculation of distances for sequence data in nexus format.

```
#include "distance_matrix.h"
#include "nexus_common.h"
```

Include dependency graph for alignment.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct alignment_struct

  Data from alignment file.

Typedefs

- typedef struct alignment_struct ∗ **alignment**

Functions

- alignment read_alignment_from_file (char ∗seqfilename)

    Reads DNA alignment (guess format between FASTA and NEXUS) from file and store info in alignment_struct.
- alignment read_fasta_alignment_from_file (char ∗seqfilename)

    Reads DNA FASTA alignment from file and store info in alignment_struct.
- alignment read_nexus_alignment_from_file (char ∗seqfilename)

    Reads DNA NEXUS alignment from file and store info in alignment_struct.
- void print_alignment_in_fasta_format (alignment align, FILE ∗stream)

    Prints alignment to FILE stream in FASTA format (debug purposes).
- void del_alignment (alignment align)

    Frees memory from alignment_struct.
- distance_matrix new_distance_matrix_from_valid_matrix_elems (distance_matrix original, int ∗valid, int n↩_valid)

    new matrix of pairwise distance by simply excluding original elements not present in valid[]
- distance_matrix new_distance_matrix_from_alignment (alignment align)

    creates and calculates matrix of pairwise distances based on alignment
- void store_likelihood_info_at_leaf (double ∗∗l, char ∗align, int n_pat, int n_state)

    transform aligned sequence into likelihood for terminal taxa (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc) (e.g. A -> 0001, C-> 0010 etc)

4.1.1 Detailed Description

File handling functions and calculation of distances for sequence data in nexus format.

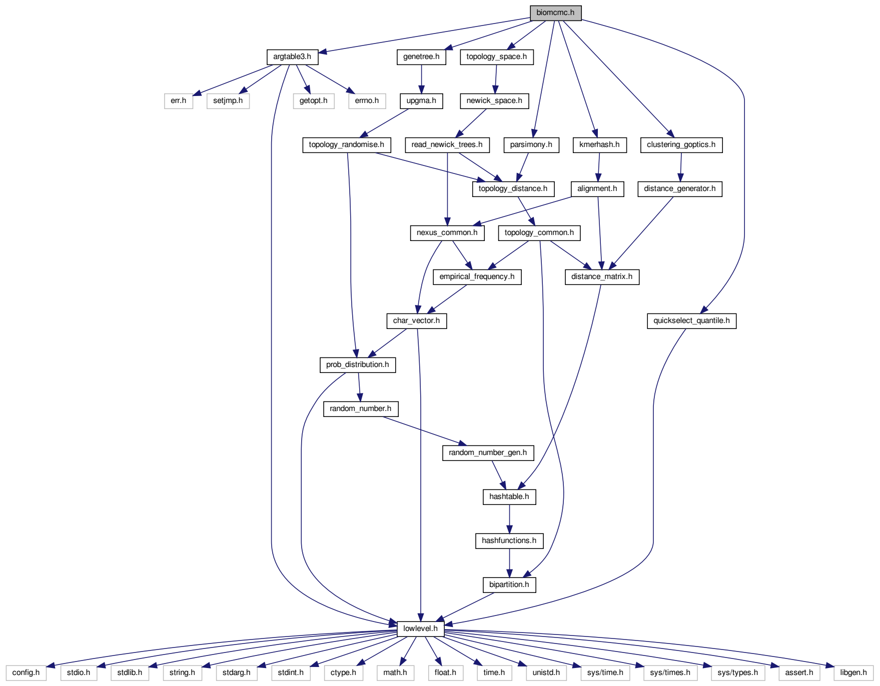
Reading of sequence data in nexus format (sequencial or interleaved) and fasta format. For fasta format the sequences don't need to be aligned, but for all formats if the sequences are aligned a data compression is used so that we keep only the distinct site (column) patterns and a mapping between original and compressed site columns. Based on the sequence pairs we can also calculate the matrix of distances between sequences.

4.2 biomcmc.h File Reference

biomcmc library interface to external programs, specific to super_sptree repo.

```
#include "argtable3.h"
#include "kmerhash.h"
#include "parsimony.h"
#include "genetree.h"
#include "topology_space.h"
#include "clustering_goptics.h"
```

`#include "quickselect_quantile.h"`
Include dependency graph for biomcmc.h:



4.2.1 Detailed Description

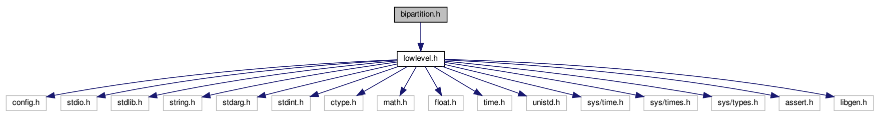
biomcmc library interface to external programs, specific to super_sptree repo.

The idea is for biomcmc-lib is to be general for several sofware, including treesignal and super_sptree. This library started branching from the biomcmc library from the guenomu software. It includes the edlib library for sequence pairwise edit distance (`http://martinsos.github.io/edlib`)

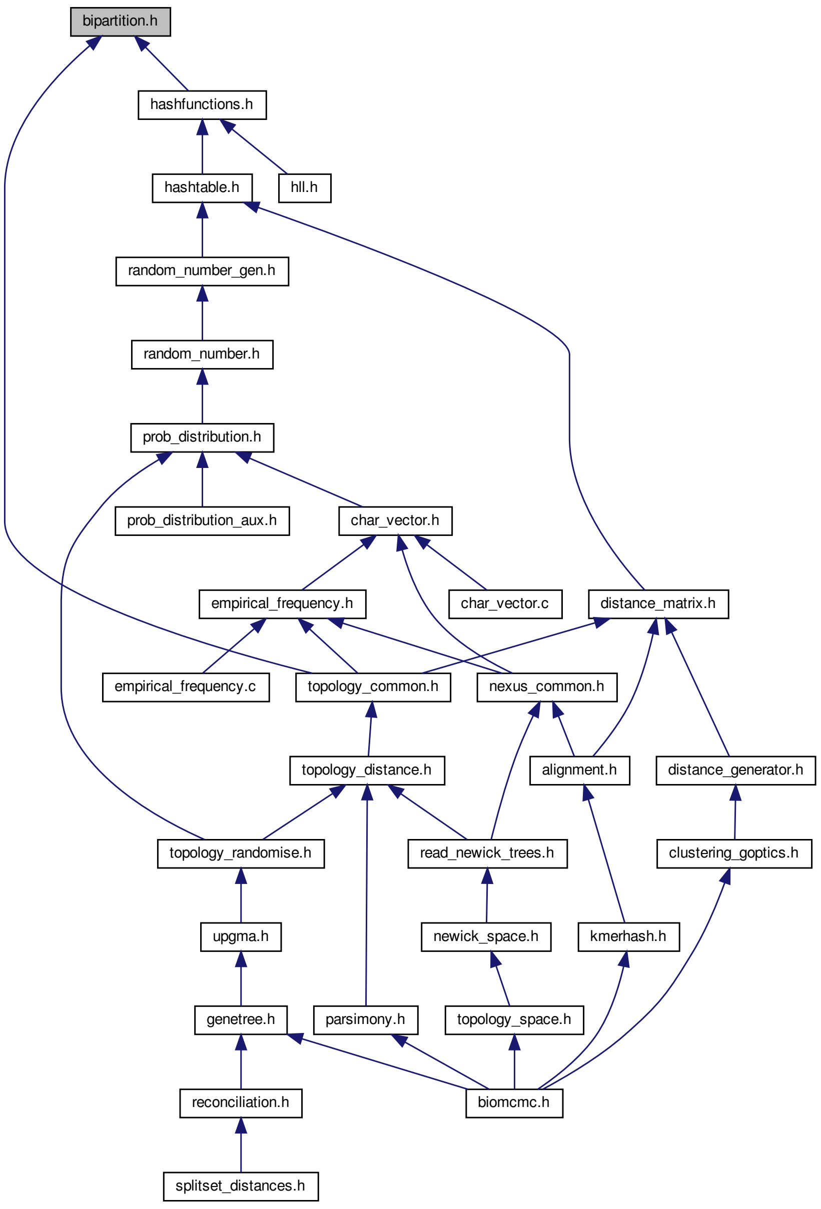
4.3 bipartition.h File Reference

Unary/binary operators on arbitrarily-sized bitstrings (strings of zeros and ones) like split bipartitions.

`#include "lowlevel.h"`
Include dependency graph for bipartition.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct bipartition_struct

  Bit-string representation of splits.
- struct bipsize_struct

Typedefs

- typedef struct bipartition_struct ∗ **bipartition**
- typedef struct bipsize_struct ∗ **bipsize**
- typedef bipartition ∗ **tripartition**

Functions

- bipartition new_bipartition (int size)

  create a new bipartition (bitstring) capable of storing an arbitrary number of bits and initialize it to zero
- bipsize new_bipsize (int size)

  Create a new bipsize, which controls some bipartition sizes.
- bipartition new_bipartition_copy_from (const bipartition from)

  Create a new bipartition (allocate memory) and initialize it from another bipartition.
- bipartition new_bipartition_from_bipsize (bipsize n)

  create new bipartition that will share bipsize – useful for bipartition vectors
- void del_bipartition (bipartition bip)

  free memory allocated by bipartition
- void del_bipsize (bipsize n)

  free memory allocated by bipsize
- void bipsize_resize (bipsize n, int nbits)

  update the valid number of bits and mask – e.g. when replacing subtrees by leaves in reduced trees
- void bipartition_initialize (bipartition bip, int position)

  set all bits to zero except the one at position-th bit
- void bipartition_zero (bipartition bip)

  set all bits to zero
- void bipartition_set (bipartition bip, int position)

  simply set the bit at "position" to one, irrespective of other bits
- void **bipartition_set_lowlevel** (bipartition bip, int i, int j)
- void bipartition_unset (bipartition bip, int position)

  simply unset the bit at "position" (set to zero), irrespective of other bits
- void **bipartition_unset_lowlevel** (bipartition bip, int i, int j)
- void bipartition_copy (bipartition to, const bipartition from)

  Copy contents from one bipartition to another.
- void bipartition_OR (bipartition result, const bipartition b1, const bipartition b2, bool update_count)

  Binary logical OR ("|") between b1 and b2, where update_count should be true if you need to know the resulting size (slow) or false if you don't care or if b1 and b2 are disjoint (no common elements)
- void bipartition_AND (bipartition result, const bipartition b1, const bipartition b2, bool update_count)

  *Binary logical AND ("&") between b1 and b2, update_count should be set to false only if you **really** don't need to know the number of active bits (e.g. sorting, bipartition comparison)*
- void bipartition_ANDNOT (bipartition result, const bipartition b1, const bipartition b2, bool update_count)

  Binary logical AND ("&") between b1 and ∼b2 (NOT b2), that is, apply mask b1 on the inverse of b2.
- void bipartition_XOR (bipartition result, const bipartition b1, const bipartition b2, bool update_count)

  *Binary logical eXclusive OR ("^") between b1 and b2, update_count should be set to false only if you **really** don't need to know the number of active bits (e.g. sorting, bipartition comparison)*
- void bipartition_XORNOT (bipartition result, const bipartition b1, const bipartition b2, bool update_count)

  Binary logical eXclusive OR ("^") between b1 and complement of b2 (that is, NOT b2: b1 ^ ∼b2). Used when finding best disagreement (that in this case erases the complement –other side – of agreement edge)
- void bipartition_NOT (bipartition result, const bipartition bip)

  Unary complement ("∼") of bipartition. Use with caution, since there is no mask for unused padded bits.
- void bipartition_count_n_ones (const bipartition bip)

  Count the number of active bits (equal to one). Used by bipartition_AND() and bipartition_XOR() when update_count = true. Please use it parsimoniously since it is as slow as without bitstring representation.
- void bipartition_to_int_vector (const bipartition b, int ∗id, int vecsize)

  fill vector id[] with positions of set bits, up to vecsize bits set
- bool bipartition_is_equal (const bipartition b1, const bipartition b2)

  Compare equality of two bipartitions.
- bool bipartition_is_equal_bothsides (const bipartition b1, const bipartition b2)

Compare if two bipartitions represent the same splits (or they are equal or one is the complement of the other)
- int compare_bipartitions_increasing (const void ∗a1, const void ∗a2)

  Bipartitions comparison, to be used by sort() since returns integer and uses (void)
- int compare_bipartitions_decreasing (const void ∗a1, const void ∗a2)

  Bipartitions comparison, to be used by sort() since returns integer and uses (void)
- bool bipartition_is_larger (const bipartition b1, const bipartition b2)

  Compare sizes of two bipartitions, by number of active bits with ties broken by actual bitstrings.
- void bipartition_flip_to_smaller_set (bipartition bip)

  invert ones and zeroes in loco when necessary to assure bipartition has more zeroes than ones
- bool bipartition_is_bit_set (const bipartition bip, int position)

  Check if position-th bit is equal to one or not.
- bool bipartition_contains_bits (const bipartition b1, const bipartition b2)

  Check if first bipartition contains all elements of second bipartition (b2 is a subset of b1)
- void bipartition_print_to_stdout (const bipartition b1)

  Print to screen a bit representation of the bipartition (with number of ones at the end)
- void bipartition_replace_bit_in_vector (bipartition ∗bvec, int n_b, int to, int from, bool reduce)

  replace bit info, copying 'from' one position 'to' another; bool "update" indicates if afterwards size will be reduced
- void bipartition_resize_vector (bipartition ∗bvec, int n_b)

  apply mask to last element (useful after manipulations) and count number of bits
- tripartition new_tripartition (int nleaves)

  tripartition of a node (a vector with 3 bipartitions, that should not be 'flipped' to smaller set, however)
- void del_tripartition (tripartition trip)

  free tripartition space (just 3 bipartitions)
- void store_tripartition_from_bipartitions (tripartition tri, bipartition b1, bipartition b2)

  from node, create tripartition from node->left and node->right (assuming bipartitions were not 'flipped' yet)
- void sort_tripartition (tripartition tri)

  sort order of bipartitions s.t. smallest is first
- int align_tripartitions (tripartition tp1, tripartition tp2, hungarian h)

  match bipartitions between two nodes and return optimal score (min disagreement)
- bool tripartition_is_equal (tripartition tp1, tripartition tp2)

  assuming tripartitions are ordered, check if nodes (represented by tripartitions) are the same

4.3.1 Detailed Description

Unary/binary operators on arbitrarily-sized bitstrings (strings of zeros and ones) like split bipartitions.

4.3.2 Function Documentation

4.3.2.1 new_bipartition()

```
bipartition new_bipartition (
              int size )
```

create a new bipartition (bitstring) capable of storing an arbitrary number of bits and initialize it to zero

Parameters

in	*size*	number of bits of desired bipartition

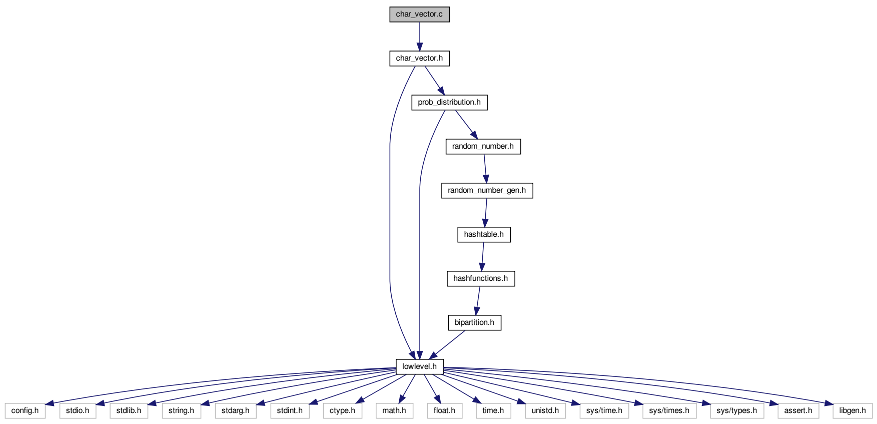
Returns

bipartition (opaquely a vector of long long ints)

4.4 char_vector.c File Reference

vector of strings (species names, leaf names, etc.)

`#include "char_vector.h"`
Include dependency graph for char_vector.c:



Data Structures

- struct charvec_str

Macros

- #define **kroundup32**(x) (--(x), (x)|=(x)>>1, (x)|=(x)>>2, (x)|=(x)>>4, (x)|=(x)>>8, (x)|=(x)>>16, ++(x))

Functions

- int **compare_charvecstr_decreasing** (const void ∗a, const void ∗b)
- int **compare_charvecstr_lexicographic** (const void ∗a, const void ∗b)
- char_vector new_char_vector (int nstrings)

    Create a vector of strings with initial size for each string of zero.
- char_vector new_char_vector_big (int nstrings)

    Create vector of strings, and preparing it to realloc() fewer times; used in conjunction with 'append_big'.
- char_vector new_char_vector_from_valid_strings_char_vector (char_vector vec, int ∗valid, int n_valid)

    Create a vector of strings from subset of strings of another char_vector.
- char_vector new_char_vector_fixed_length (int nstrings, int nchars)

Create a vector of strings where each string is assigned an initial value of nchars.

- void del_char_vector (char_vector vec)

  Delete vector of strings only after nobody is using it.

- void char_vector_link_string_at_position (char_vector vec, const char ∗string, int position)

  Link a previously allocated string (to avoid copying all characters)

- void char_vector_add_string_at_position (char_vector vec, const char ∗string, int position)

  Add a new string (vector of characters) at specific location.

- void char_vector_add_string (char_vector vec, const char ∗string)

  Add a new string (vector of characters) at next available location.

- void char_vector_append_string_at_position (char_vector vec, const char ∗string, int position)

  Append string at the end of existing string at location.

- void char_vector_append_string (char_vector vec, const char ∗string)

  Append string at the end of existing string at most recently used location.

- void char_vector_append_string_big_at_position (char_vector vec, const char ∗string, int position)

  Append strings like before, but doubling allocation space if insufficient (reduces calls to realloc())

- void **char_vector_append_string_big** (char_vector vec, const char ∗string)
- void **char_vector_finalise_big** (char_vector vec)
- void char_vector_expand_nstrings (char_vector vec, int new_size)

  Increase size of vector of strings (called automatically by other functions)

- void char_vector_reorder_strings_from_external_order (char_vector vec, int ∗order)

  update order of strings in vector based on a vector of new positions

- int char_vector_remove_empty_strings (char_vector vec)

  Reduce size of vector of strings by removing empty strings (returns number of empty strings)

- int char_vector_remove_duplicate_strings (char_vector vec)

  Remove identical strings and resizes char_vector_struct.

- void char_vector_reduce_to_valid_strings (char_vector vec, int ∗valid, int n_valid)

  reduce char_string_struct to only those elements indexed by valid[]

- void char_vector_reorder_by_size_or_lexicographically (char_vector vec, bool lexico, int ∗order)

  Order char_vector_struct elements from longer string to smaller, or lexicographically; can be used after calling new←_char_vector_from_file() but if topology etc. are associated to it, then order[] must be externally defined and will have new locations, to keep track of changes.

- bool char_vector_link_address_if_identical (char_vector ∗v1, char_vector ∗v2)

  If the two char_vectors are identical (same strings in same order), then delete one and make it point to the other one.

- void index_species_gene_char_vectors (char_vector species, char_vector gene, int ∗sp_idx_in_gene, int ∗order_external)

  find occurences of species->string[] inside gene->string[] filling indexes in sp_idx_in_gene.

- void **update_species_count_from_gene_char_vector** (char_vector species, char_vector gene, int ∗sp_←count)

4.4.1 Detailed Description

vector of strings (species names, leaf names, etc.)

4.4.2 Function Documentation

4.4.2.1 index_species_gene_char_vectors()

```
void index_species_gene_char_vectors (
            char_vector species,
            char_vector gene,
            int * sp_idx_in_gene,
            int * order_external )
```

find occurences of species->string[] inside gene->string[] filling indexes in sp_idx_in_gene.

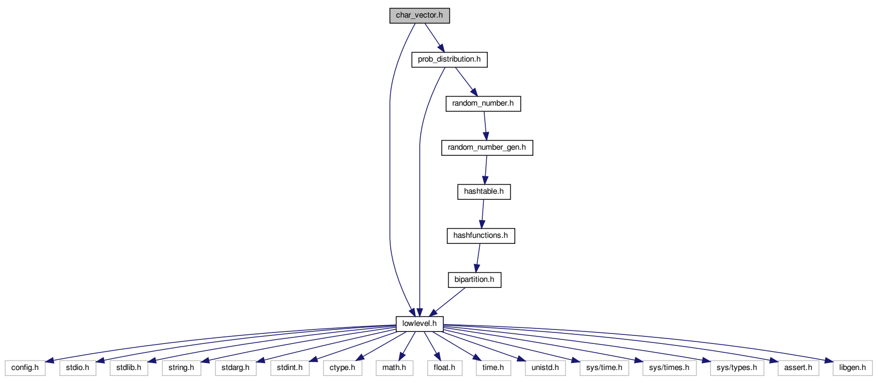
The species are taxon names which may be associated with topologies or alignments, such that we can not reorder its elements here (without also modifing e.g. tree leaves). But ordering from longer to shorter is essential for pattern finding, so it is assumed that the char_vector is already sorted UNLESS user provides the ordering.

4.5 char_vector.h File Reference

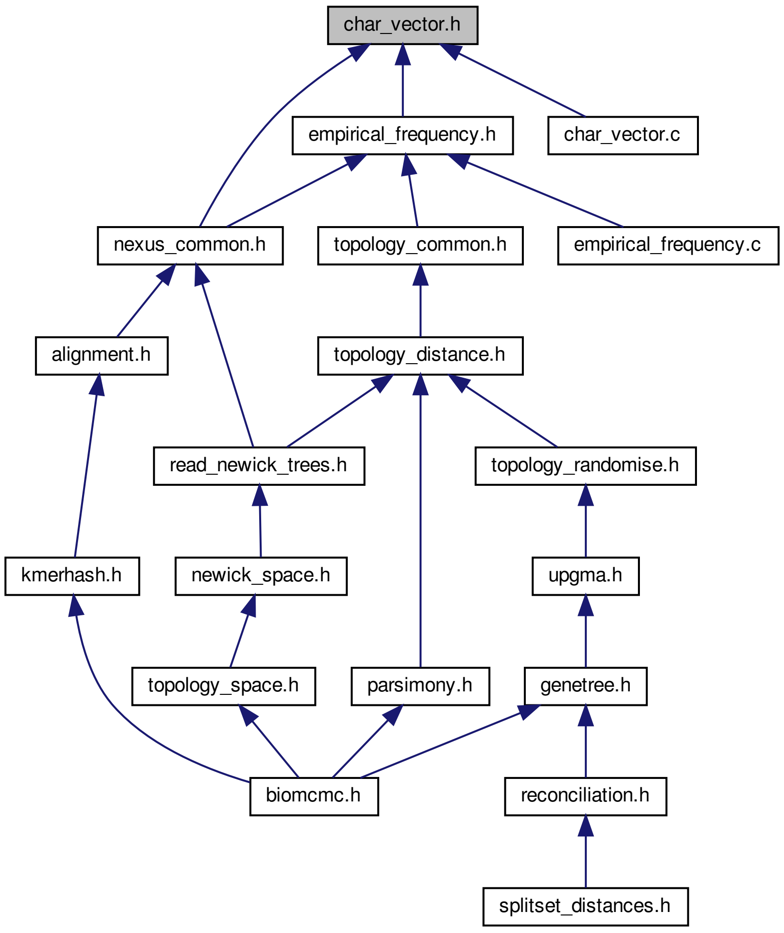
list of strings (each string is a vector of chars)

```
#include "lowlevel.h"
#include "prob_distribution.h"
```

Include dependency graph for char_vector.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct char_vector_struct

    vector of strings (char vectors) of variable length

Typedefs

- typedef struct char_vector_struct ∗ **char_vector**

Functions

- char_vector new_char_vector (int nstrings)

    Create a vector of strings with initial size for each string of zero.
- char_vector new_char_vector_big (int nstrings)

    Create vector of strings, and preparing it to realloc() fewer times; used in conjunction with 'append_big'.
- char_vector new_char_vector_from_valid_strings_char_vector (char_vector vec, int ∗valid, int n_valid)

Create a vector of strings from subset of strings of another char_vector.
- char_vector new_char_vector_fixed_length (int nstrings, int nchars)

  Create a vector of strings where each string is assigned an initial value of nchars.
- void del_char_vector (char_vector vec)

  Delete vector of strings only after nobody is using it.
- void char_vector_link_string_at_position (char_vector vec, const char ∗string, int position)

  Link a previously allocated string (to avoid copying all characters)
- void char_vector_add_string_at_position (char_vector vec, const char ∗string, int position)

  Add a new string (vector of characters) at specific location.
- void char_vector_add_string (char_vector vec, const char ∗string)

  Add a new string (vector of characters) at next available location.
- void char_vector_append_string_at_position (char_vector vec, const char ∗string, int position)

  Append string at the end of existing string at location.
- void char_vector_append_string (char_vector vec, const char ∗string)

  Append string at the end of existing string at most recently used location.
- void char_vector_append_string_big_at_position (char_vector vec, const char ∗string, int position)

  Append strings like before, but doubling allocation space if insufficient (reduces calls to realloc())
- void **char_vector_append_string_big** (char_vector vec, const char ∗string)
- void **char_vector_finalise_big** (char_vector vec)
- void char_vector_expand_nstrings (char_vector vec, int new_size)

  Increase size of vector of strings (called automatically by other functions)
- void char_vector_reorder_strings_from_external_order (char_vector vec, int ∗order)

  update order of strings in vector based on a vector of new positions
- int char_vector_remove_empty_strings (char_vector vec)

  Reduce size of vector of strings by removing empty strings (returns number of empty strings)
- int char_vector_remove_duplicate_strings (char_vector vec)

  Remove identical strings and resizes char_vector_struct.
- void char_vector_reduce_to_valid_strings (char_vector vec, int ∗valid, int n_valid)

  reduce char_string_struct to only those elements indexed by valid[]
- void char_vector_reorder_by_size_or_lexicographically (char_vector vec, bool lexico, int ∗order)

  Order char_vector_struct elements from longer string to smaller, or lexicographically; can be used after calling new←↩_char_vector_from_file() but if topology etc. are associated to it, then order[] must be externally defined and will have new locations, to keep track of changes.
- bool char_vector_link_address_if_identical (char_vector ∗v1, char_vector ∗v2)

  If the two char_vectors are identical (same strings in same order), then delete one and make it point to the other one.
- void index_species_gene_char_vectors (char_vector species, char_vector gene, int ∗sp_idx_in_gene, int ∗order_external)

  find occurences of species->string[] inside gene->string[] filling indexes in sp_idx_in_gene.
- void **update_species_count_from_gene_char_vector** (char_vector species, char_vector gene, int ∗sp_←↩count)

4.5.1 Detailed Description

list of strings (each string is a vector of chars)

4.5.2 Function Documentation

4.5.2.1 index_species_gene_char_vectors()

```
void index_species_gene_char_vectors (
            char_vector species,
            char_vector gene,
            int * sp_idx_in_gene,
            int * order_external )
```

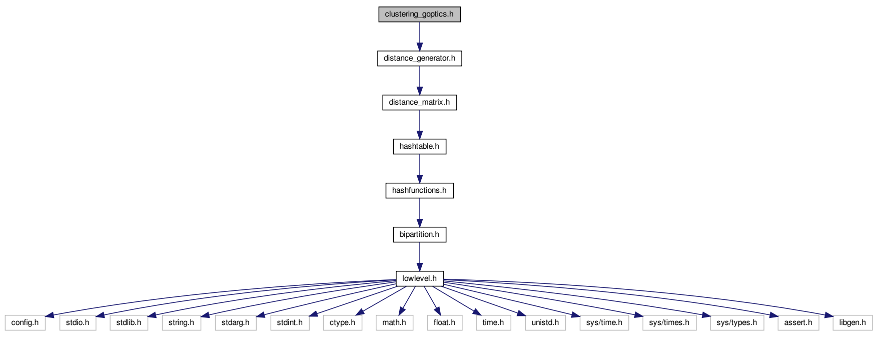
find occurences of species->string[] inside gene->string[] filling indexes in sp_idx_in_gene.

The species are taxon names which may be associated with topologies or alignments, such that we can not reorder its elements here (without also modifing e.g. tree leaves). But ordering from longer to shorter is essential for pattern finding, so it is assumed that the char_vector is already sorted UNLESS user provides the ordering.

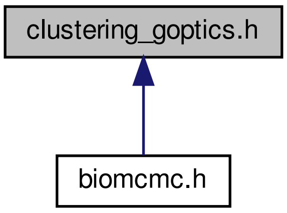
4.6 clustering_goptics.h File Reference

OPTICS algorithm based on https://github.com/guineri/GOPTICS.

`#include "distance_generator.h"`
Include dependency graph for clustering_goptics.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct goptics_cluster_struct

Typedefs

- typedef struct goptics_cluster_struct ∗ **goptics_cluster**

Functions

- goptics_cluster **new_goptics_cluster** (distance_generator dg, int min_points, double epsilon)
- goptics_cluster **new_goptics_cluster_run** (distance_generator dg, int min_points, double epsilon)
- void **del_goptics_cluster** (goptics_cluster gop)
- void **assign_goptics_clusters** (goptics_cluster gop, double cluster_eps)

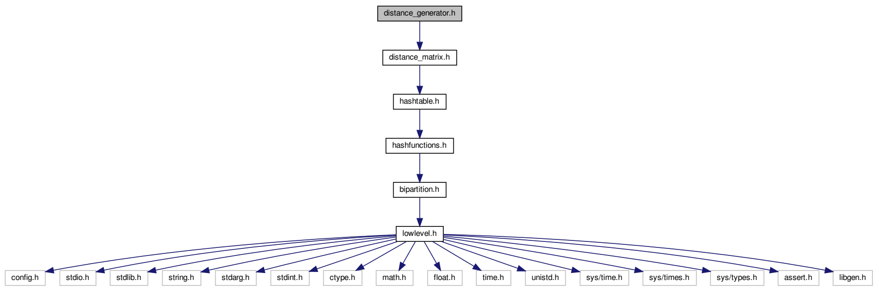
4.6.1 Detailed Description

OPTICS algorithm based on `https://github.com/guineri/GOPTICS`.

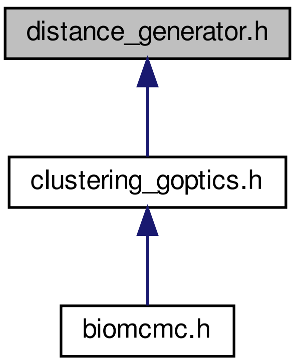
4.7 distance_generator.h File Reference

distance calculation between generic objects,without generating full matrix beforehand

```
#include "distance_matrix.h"
```
Include dependency graph for distance_generator.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct distance_generator_struct

Typedefs

- typedef struct distance_generator_struct ∗ **distance_generator**

Functions

- distance_generator **new_distance_generator** (int n_samples, int n_distances)
- void **del_distance_generator** (distance_generator d)
- double **distance_generator_get_at_distance** (distance_generator d, int i, int j, int which_distance)
- double **distance_generator_get** (distance_generator d, int i, int j)
- void distance_generator_set_function_data (distance_generator d, void(∗lowlevel_dist_funct)(void ∗, int, int, double ∗), void ∗extra_data)

  defines distance calculation function wrapper, and all extra data needed by wrapper; no check is done here, but wrapper should return at least as many distances sd n_distances (wrapper functions can check)
- void distance_generator_set_which_distance (distance_generator d, int which_distance)

  distance wrapper may return several distances, but only one is returned by get(); this sets which one (should be called before e.g. clustering)
- void **distance_generator_reset** (distance_generator d)

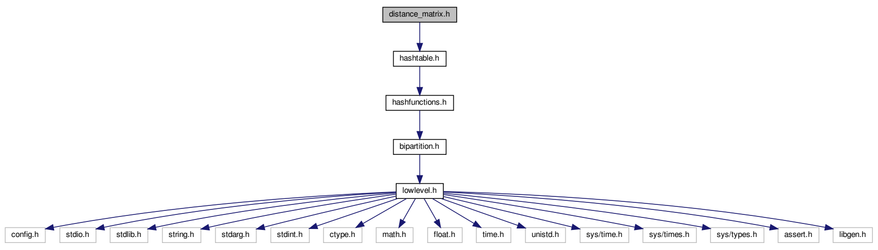
4.7.1 Detailed Description

distance calculation between generic objects,without generating full matrix beforehand

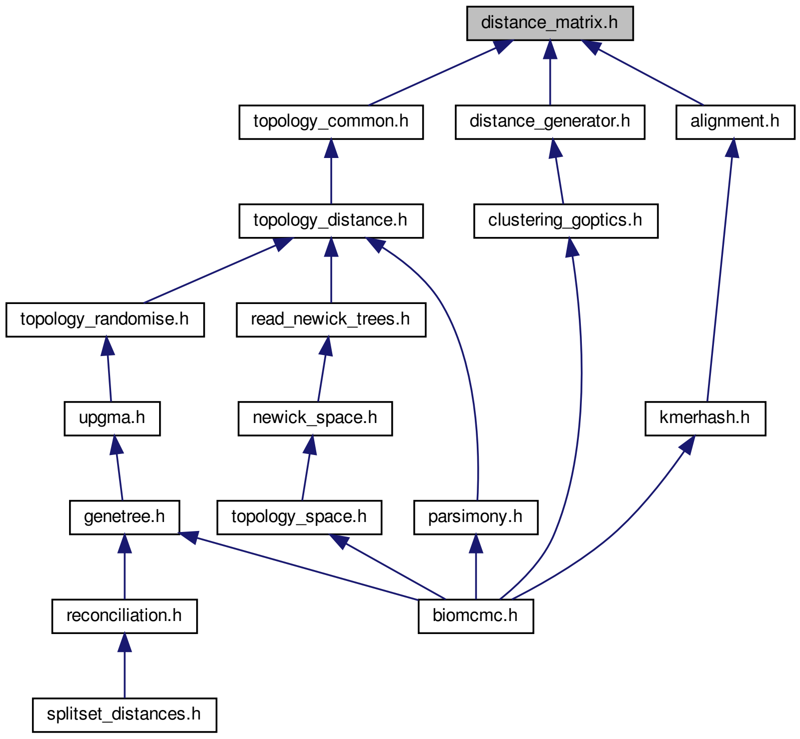
4.8 distance_matrix.h File Reference

distance matrix, that can be used in alignments and trees, and patristic-distance based species distances

```
#include "hashtable.h"
```
Include dependency graph for distance_matrix.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct distance_matrix_struct
- struct spdist_matrix_struct

Typedefs

- typedef struct distance_matrix_struct ∗ **distance_matrix**
- typedef struct spdist_matrix_struct ∗ **spdist_matrix**

Functions

- distance_matrix new_distance_matrix (int nseqs)

  creates new matrix of pairwise distances
- void zero_lower_distance_matrix (distance_matrix dist)

  specially in gene/sptree distance methods (GLASS, STEAC, etc.) lower is used for means and upper for min. This function resets matrix elements
- void transpose_distance_matrix (distance_matrix dist)

  invert lower and upper diagonals of matrix (since some functions like upgma expect upper, etc.)
- void del_distance_matrix (distance_matrix dist)

  releases memory allocated to distance_matrix (this structure has no smart ref_counter)
- spdist_matrix **new_spdist_matrix** (int n_species)

- void zero_all_spdist_matrix (spdist_matrix dist)
- void **finalise_spdist_matrix** (spdist_matrix dist)
- void **finalise_spdist_matrix_with_rescaling** (spdist_matrix dist, double scale)
- void **complete_missing_spdist_from_global_spdist** (spdist_matrix local, spdist_matrix global)
- void **copy_spdist_matrix_to_distance_matrix_upper** (spdist_matrix spd, distance_matrix dist, bool use←_means)
- void **del_spdist_matrix** (spdist_matrix dist)
- void fill_species_dists_from_gene_dists (distance_matrix spdist, distance_matrix gendist, int ∗sp_id, bool use_upper_gene)

  updates distances between species based on genes and gene-to-species mapping, with min on upper and mean on lower diagonal
- void update_species_dists_from_spdist (distance_matrix global, distance_matrix local, int ∗spexist)

  update global (over loci) species distances besed on local (within locus) species distances
- int **prepare_spdistmatrix_from_gene_species_map** (spdist_matrix spdist, int ∗sp_id, int n_sp_id)
- void **fill_spdistmatrix_from_gene_dists** (spdist_matrix spdist, distance_matrix gendist, int ∗sp_id, bool use_upper_gene)
- void fill_spdistmatrix_from_gene_dist_vector (spdist_matrix spdist, double ∗gdist, int n_gdist, int ∗sp_id)

  initialise spdist_matrix with patristic distances from gdist vector of size n_gdist (1D)
- void **update_spdistmatrix_from_spdistmatrix** (spdist_matrix global, spdist_matrix local)

4.8.1 Detailed Description

distance matrix, that can be used in alignments and trees, and patristic-distance based species distances

These functions don't know about trees/topologies: topology_common.c creates the actual patristic distances, and downstream software or genetree.h should decide how to use this information

4.8.2 Function Documentation

4.8.2.1 zero_all_spdist_matrix()

```
void zero_all_spdist_matrix (
            spdist_matrix dist )
```

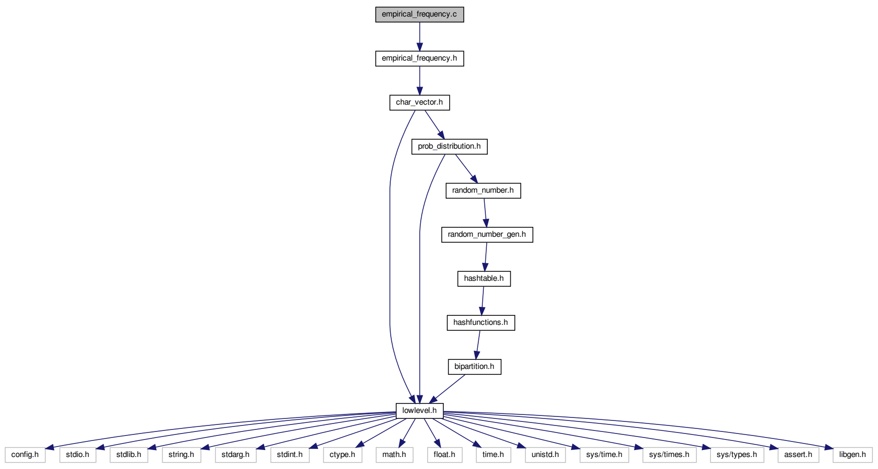
zero both mean[] and min[] since we only look at average (never min) across loci

4.9 empirical_frequency.c File Reference

histogram of vectors, ordered by frequency. Also calculates MAP (modal) values.

```
#include "empirical_frequency.h"
```

Include dependency graph for empirical_frequency.c:



Functions

- int **compare_empfreq_element_decreasing** (const void ∗a, const void ∗b)
- int **compare_empfreq_element_increasing** (const void ∗a, const void ∗b)
- int **compare_empfreq_double_element_decreasing** (const void ∗a, const void ∗b)
- int **compare_empfreq_double_element_increasing** (const void ∗a, const void ∗b)
- empfreq **create_empfreq_from_value_sorted_empfreq** (empfreq e_idx)
- void **sort_empfreq_decreasing** (empfreq ef)
- void **sort_empfreq_increasing** (empfreq ef)
- void **sort_empfreq_double_decreasing** (empfreq_double efd)
- void **sort_empfreq_double_increasing** (empfreq_double efd)
- empfreq **new_empfreq** (int n_elements)
- void **del_empfreq** (empfreq ef)
- empfreq_double **new_empfreq_double** (int n_elements)
- void **del_empfreq_double** (empfreq_double efd)
- empfreq **new_empfreq_sort_decreasing** (void ∗vec, int n, char type)
- empfreq **new_empfreq_sort_increasing** (void ∗vec, int n, char type)
- empfreq_double **new_empfreq_double_sort_decreasing** (double ∗vec, int n)
- empfreq_double **new_empfreq_double_sort_increasing** (double ∗vec, int n)
- empfreq **new_empfreq_from_int** (int ∗vec, int n)
- empfreq **new_empfreq_from_int_weighted** (int ∗vec, int n, int ∗weight)
- int **find_mode_int** (int ∗vec, int n)
- int **find_mode_int_weighted** (int ∗vec, int n, int ∗weight)

4.9.1 Detailed Description

histogram of vectors, ordered by frequency. Also calculates MAP (modal) values.

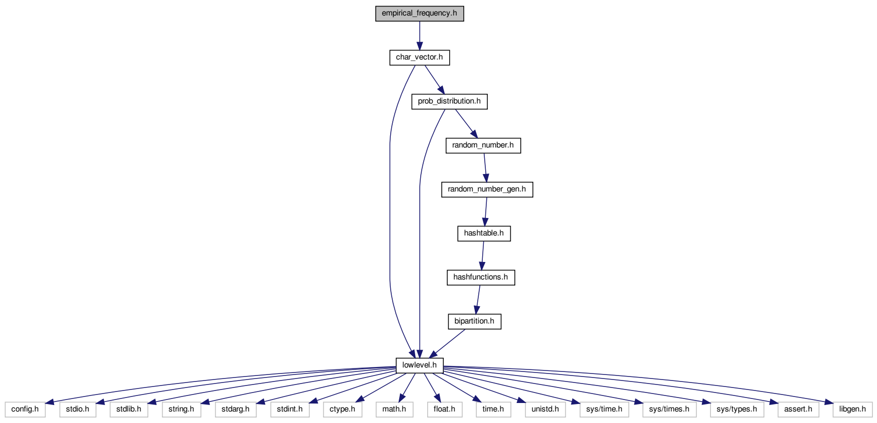
Sorts a vector of integers by their frequencies, preserving their original indexes. It is a simple extension to qsort where the original order can be reconstructed, or still a key/value sorting.

4.10 empirical_frequency.h File Reference

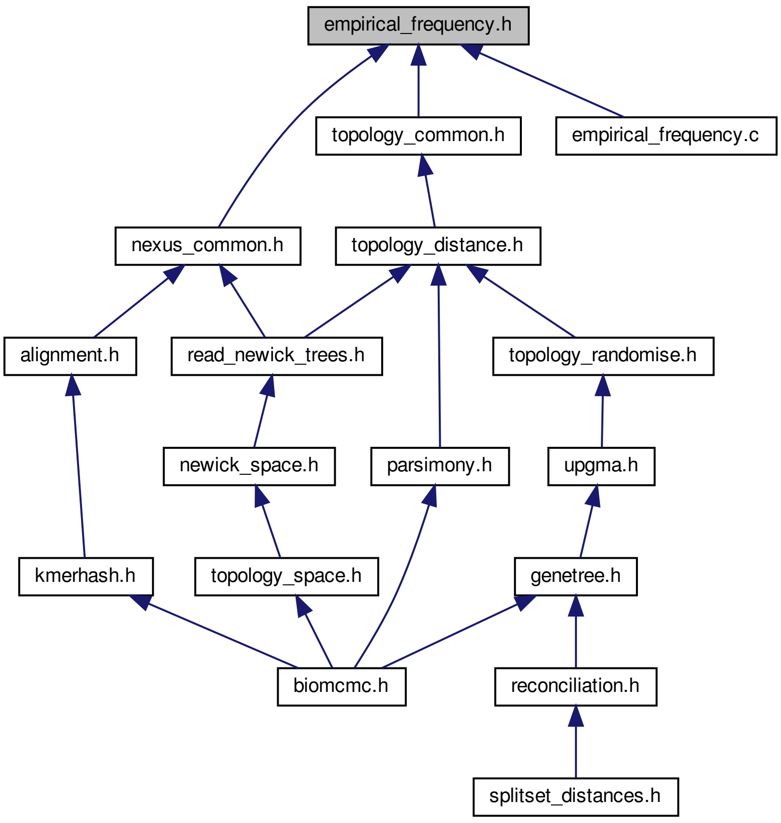
Creates a histogram of a vector, ordered by frequency.

```
#include "char_vector.h"
```

Include dependency graph for empirical_frequency.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct empfreq_element
- struct empfreq_double_element
- struct empfreq_struct
- struct empfreq_double_struct

Typedefs

- typedef struct empfreq_struct ∗ **empfreq**
- typedef struct empfreq_double_struct ∗ **empfreq_double**

Functions

- void **sort_empfreq_decreasing** (empfreq ef)
- void **sort_empfreq_increasing** (empfreq ef)
- void **sort_empfreq_double_decreasing** (empfreq_double efd)
- void **sort_empfreq_double_increasing** (empfreq_double efd)
- empfreq **new_empfreq** (int n_elements)
- void **del_empfreq** (empfreq ef)
- empfreq_double **new_empfreq_double** (int n_elements)
- void **del_empfreq_double** (empfreq_double efd)
- empfreq **new_empfreq_sort_decreasing** (void ∗vec, int n, char type)
- empfreq **new_empfreq_sort_increasing** (void ∗vec, int n, char type)
- empfreq_double **new_empfreq_double_sort_decreasing** (double ∗vec, int n)
- empfreq_double **new_empfreq_double_sort_increasing** (double ∗vec, int n)
- empfreq **new_empfreq_from_int** (int ∗vec, int n)
- empfreq **new_empfreq_from_int_weighted** (int ∗vec, int n, int ∗weight)
- int **find_mode_int** (int ∗vec, int n)
- int **find_mode_int_weighted** (int ∗vec, int n, int ∗weight)

4.10.1 Detailed Description

Creates a histogram of a vector, ordered by frequency.

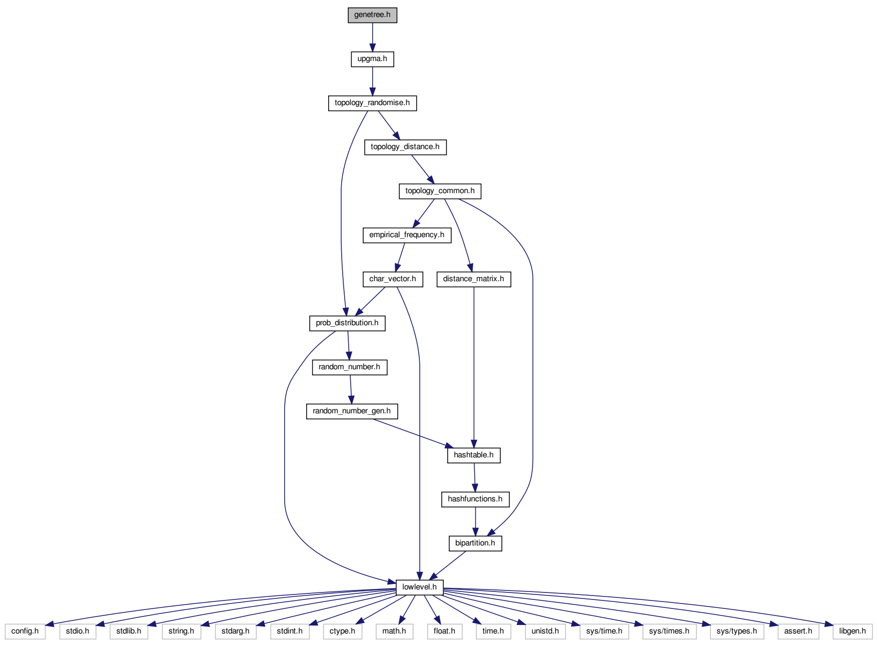
Sorts a vector of integers by their frequencies, preserving their original indexes. It is a simple extension to qsort where the original order can be reconstructed, or still a key/value sorting.

4.11 genetree.h File Reference

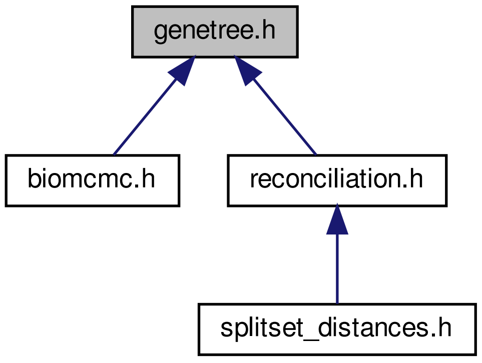
gene tree and species tree structures, for reconciliation etc. This is the high-level file with globally exposed functions/structures.

```
#include "upgma.h"
```

Include dependency graph for genetree.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct genetree_struct
- struct speciestree_struct
- struct reconciliation_struct

  mapping between gene tree nodes (this) and (external) species tree nodes
- struct splitset_struct

Typedefs

- typedef struct genetree_struct $*$ **genetree**
- typedef struct speciestree_struct $*$ **speciestree**
- typedef struct reconciliation_struct $*$ **reconciliation**
- typedef struct splitset_struct $*$ **splitset**

Functions

- genetree **new_genetree_speciestree_pair** (topology gene, topology species)
- genetree new_genetree (topology gene, speciestree sptre)

  Allocate space for new genetree_struct, given a gene topology and a specestree_struct.
- void **del_genetree** (genetree gtre)
- speciestree new_speciestree (topology species, int $*$order_of_species_names)

  Allocate space for new speciestree_struct, given a species topology and optionally the order of species names.
- void **del_speciestree** (speciestree sptre)
- void genetree_speciestree_distances (genetree gtre, speciestree sptre)

  calculates all (discrete) distances and update min and max
- int count_species_in_index_species_gene (int $*$sp_id, int max_sp, int n_sp_id)

  from gene-species map index, count number of distinct species represented
- void genetree_reconcile_speciestree (genetree gtre, speciestree sptre)

  <debug function>=""> dups.loss, ils calculation; accepts unseen speciestree_struct (i.e. updates mrca and pointers). Calls low-level hidden function.
- void genetree_dSPR_speciestree (genetree gtre, speciestree sptre, int level)

  <debug function>=""> dSPR (level $>$ 1), hdist (level $>$ 0), and RF distances; doesn't need to update sptree pointer

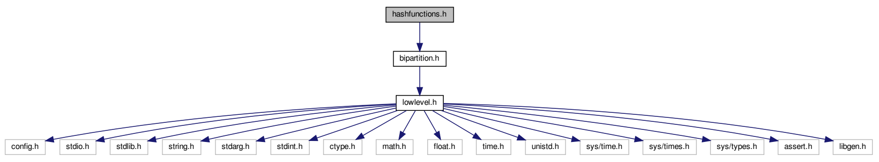
4.11.1 Detailed Description

gene tree and species tree structures, for reconciliation etc. This is the high-level file with globally exposed functions/structures.

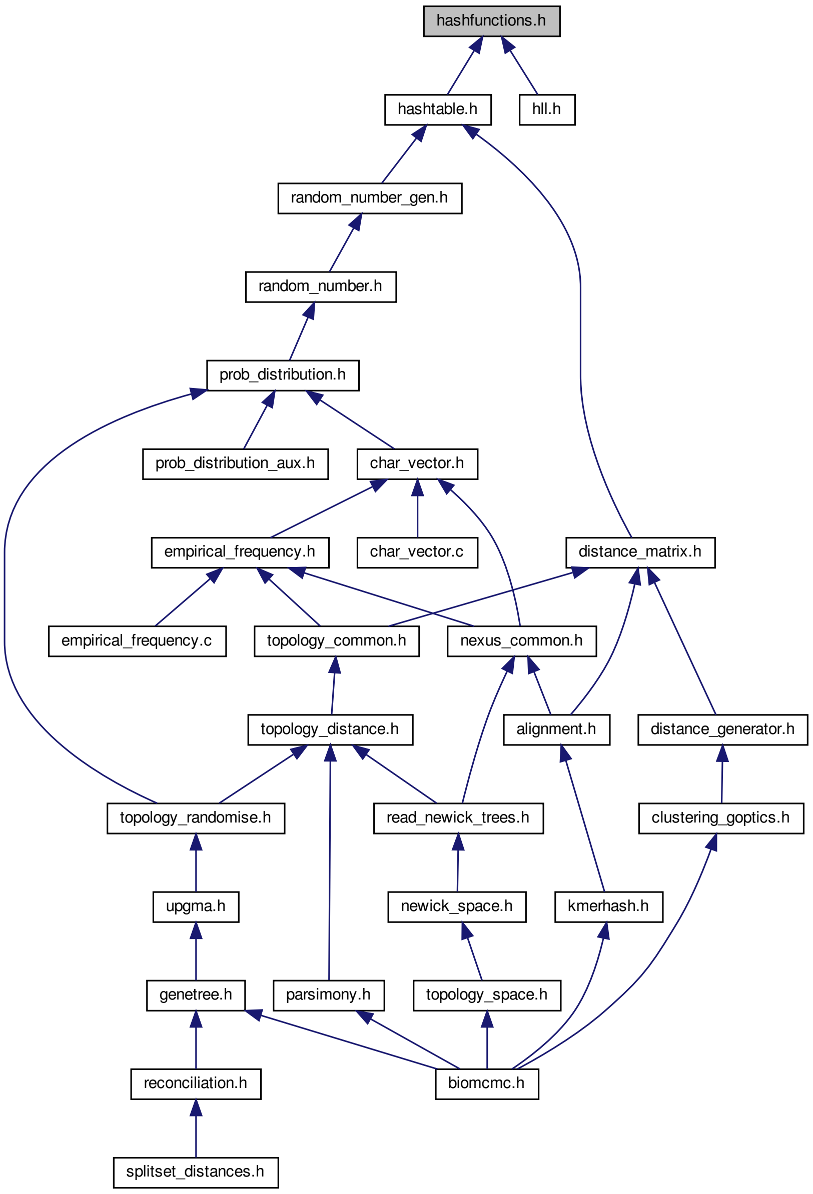
4.12 hashfunctions.h File Reference

Collections of hash functions for 32 and 64 bits, including one-liners, murmurhash, and xxhash.

```
#include "bipartition.h"
```
Include dependency graph for hashfunctions.h:



This graph shows which files directly or indirectly include this file:



Functions

- uint32_t **biomcmc_hashint_salted** (uint32_t a, int salt)
- uint32_t **biomcmc_hashbyte_salted** (const void ∗str, size_t size, int salt)
- uint64_t **biomcmc_hashint64_salted** (uint64_t k, int salt)
- uint32_t **biomcmc_hashint_mix_salted** (uint32_t a, uint32_t b, int salt)
- uint64_t **biomcmc_hashint64_mix_salted** (uint64_t a, uint64_t b, int salt)
- uint32_t **biomcmc_hashint_64to32_seed** (uint64_t x, int seed)
- uint32_t **biomcmc_hashint_64to32** (uint64_t key)
- uint32_t bipartition_hash (bipartition bip)

    32bits hash value for bipartition

- uint64_t biomcmc_murmurhash3_128bits (const void ∗key, const size_t len, const uint32_t seed, void ∗out)

  murmurhash3 using 64bits to return 128 bits (4 ints) of hash into out[] and also 64 bits as return value The 64bits is the format used internally (for speed), but the input can be a vector of any size (>1 byte)
- uint64_t biomcmc_murmurhash3_64bits (const void ∗key, const size_t len, const uint32_t seed)

  convenience function for calling mumurhash_128bits without an output vector
- uint32_t biomcmc_murmurhash3_32bits (const void ∗data, size_t nbytes, const uint32_t seed)

  murmurhash3 using 32bits to return 32 bits of hash as return value
- uint64_t biomcmc_xxh64 (const void ∗input, const size_t len, const uint32_t seed)

  xxhash function for 64 bits

4.12.1 Detailed Description

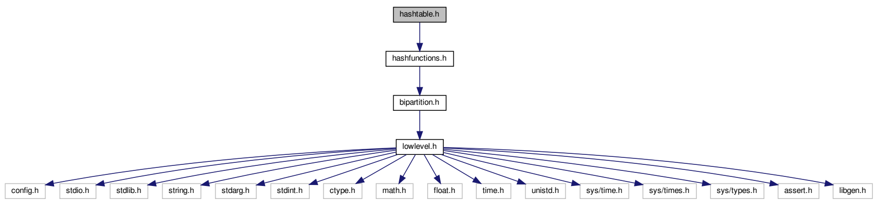
Collections of hash functions for 32 and 64 bits, including one-liners, murmurhash, and xxhash.

4.13 hashtable.h File Reference

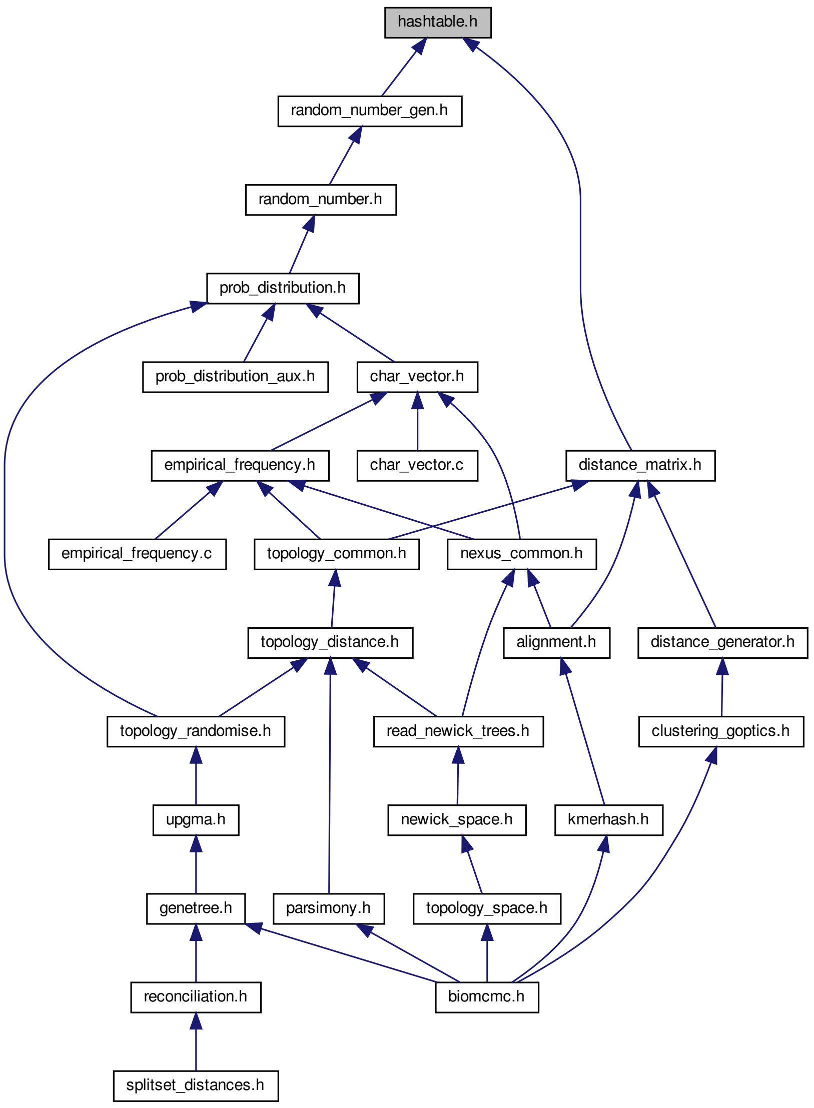
double hashing open-address hash table using strings as key – also has distance matrix, that can be used in alignments and trees

```
#include "hashfunctions.h"
```

Include dependency graph for hashtable.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct hashtable_item_struct

  key/value pair for hash table
- struct hashtable_struct

  Hash table (vector indexed by strings).
- struct bip_hashtable_struct

  Hash table of bipartitions (see hashtable.h for original version, with string keys and integer values)
- struct bip_hashitem_struct

  key (bipartition) and value (frequency) pair for hash table of bipartitions

Typedefs

- typedef struct hashtable_struct ∗ **hashtable**
- typedef struct hashtable_item_struct ∗ **hashtable_item**
- typedef struct bip_hashtable_struct ∗ **bip_hashtable**
- typedef struct bip_hashitem_struct ∗ **bip_hashitem**

Functions

- void insert_hashtable (hashtable ht, char ∗key, int value)

  Insert key/value pair into hashtable.
- int lookup_hashtable (hashtable ht, char ∗key)

  Return location (value) of corresponding key (string) or negative value if not found.
- hashtable new_hashtable (int size)

  Create new hashtable of size elements.
- void del_hashtable (hashtable ht)

  Free hashtable space.
- bip_hashtable new_bip_hashtable (int size)

  Create new hashtable of size bipartitions.
- void del_bip_hashtable (bip_hashtable ht)

  Free bipartition hashtable space.
- void bip_hashtable_insert (bip_hashtable ht, bipartition key)

  Insert key (bipartition) into bipartition hashtable, adding one to its count (freq).
- double bip_hashtable_get_frequency (bip_hashtable ht, bipartition key)

  Return frequency of bipartition (count/maxfreq) or zero if not found.

4.13.1 Detailed Description

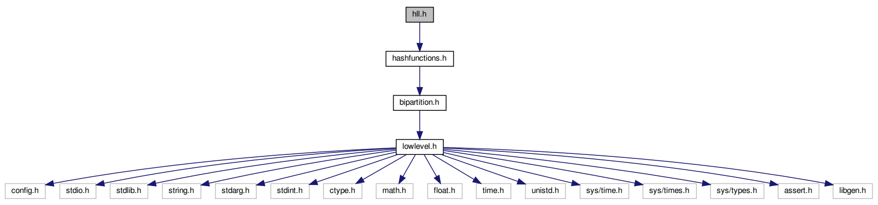
double hashing open-address hash table using strings as key – also has distance matrix, that can be used in alignments and trees

Hash tables allow us to search for the position of a key (taxa name) without scanning the whole vector (like in sequencial search). This code is derived from the software DCM3, released under the GPL license (Copyright (C) 2004 The University of Texas at Austin).

4.14 hll.h File Reference

HyperLogLog functions, based on code by Ivan Vitjuk `https://github.com/ivitjuk/libhll` under an ISC License.

`#include "hashfunctions.h"`
Include dependency graph for hll.h:



Data Structures

- struct hll_estimate_s

Typedefs

- typedef struct hll_s **hll_t**
- typedef struct hll_estimate_s hll_estimate_t
- typedef uint64_t(∗ hll_hash_function_t) (const char ∗, size_t)

Functions

- hll_t ∗ hll_create (size_t bucket_bits)
- void hll_reset (hll_t ∗hll)
- void hll_release (hll_t ∗hll)
- void hll_add (const hll_t ∗hll, const char ∗data, size_t data_len)
- int hll_merge (const hll_t ∗hll1, const hll_t ∗hll2)
- int **hll_set_hash_function** (hll_t ∗hll, hll_hash_function_t hash_function)
- int hll_get_estimate (const hll_t ∗hll, hll_estimate_t ∗estimate)

4.14.1 Detailed Description

HyperLogLog functions, based on code by Ivan Vitjuk `https://github.com/ivitjuk/libhll` under an ISC License.

4.14.2 Typedef Documentation

4.14.2.1 hll_estimate_t

```
typedef struct hll_estimate_s hll_estimate_t
```

Estimation result type

4.14.2.2 hll_hash_function_t

```
typedef uint64_t(* hll_hash_function_t) (const char *, size_t)
```

Hash function type

Even though hash funtion is expected to return a 64bit value, only 32 bits of entropy will be used.

4.14.3 Function Documentation

4.14.3.1 hll_create()

```
hll_t* hll_create (
            size_t bucket_bits )
```

Create HLL data structure

Parameters

bucket_bits	- Number of bits to use for the buckets. Actual number of buckets will be 2^bucket_bits. Must be 4 <= bucket_bits <= 16.

Returns

HLL data type or 0 on error. Error can be memory allocation failure or invalid bucket_bits value.

4.14.3.2 hll_reset()

```
void hll_reset (
            hll_t * hll )
```

Reset state of the estimator

Parameters

hll	- HLL data type

4.14.3.3 hll_release()

```
void hll_release (
            hll_t * hll )
```

Release HLL type previously allocated with hll_create().

Parameters

hll	- HLL data type

4.14.3.4 hll_add()

```
void hll_add (
            const hll_t * hll,
            const char * data,
            size_t data_len )
```

Add a sample to the HLL estimator

Parameters

hll	- HLL data type
data	- Sample to be added to the estimator (underlying data type is not important)
data_len	- Lenght of the data sample in bytes

4.14.3.5 hll_merge()

```
int hll_merge (
             const hll_t * hll1,
             const hll_t * hll2 )
```

Merge data from two HLLs

Data from hll2 will be merged into hll2

Parameters

hll1	- First HLL data type
hll2	- Second HLL data type

Returns

1 on success, 0 on failure. Fails when number of buckets are not compatible.

4.14.3.6 hll_get_estimate()

```
int hll_get_estimate (
             const hll_t * hll,
             hll_estimate_t * estimate )
```

Get the estimated cardinality based on the data added to the estimator

Parameters

hll	- HLL data type
estimate	- Result of the estimation

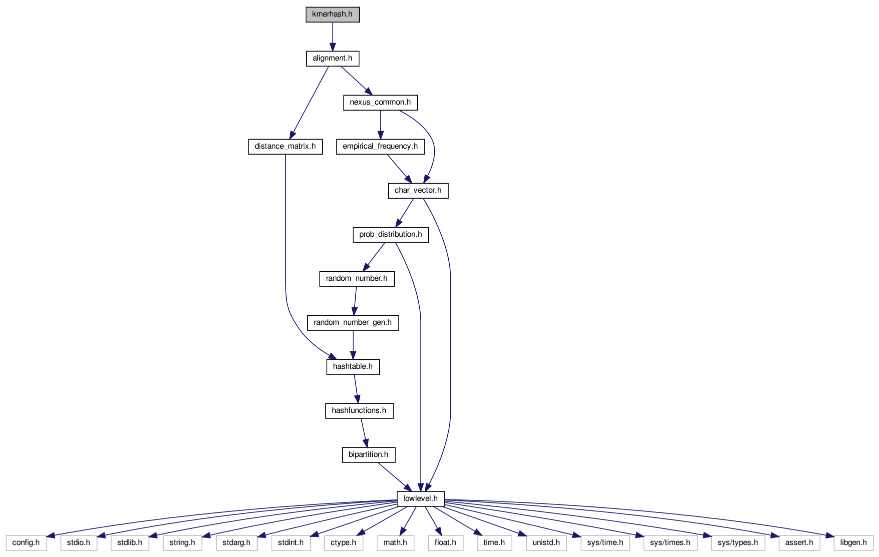
Returns

1 on success, 0 on failure. Fails only on NULL input parameters.

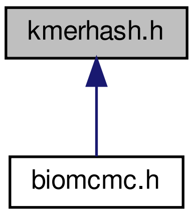
4.15 kmerhash.h File Reference

k-mer handling of DNA sequences, with hash transformation

```
#include "alignment.h"
```
Include dependency graph for kmerhash.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct kmer_params_struct
- struct kmerhash_struct

Typedefs

- typedef struct kmer_params_struct ∗ **kmer_params**
- typedef struct kmerhash_struct ∗ **kmerhash**

Functions

- kmer_params **new_kmer_params** (int mode)
- void **del_kmer_params** (kmer_params p)
- kmerhash **new_kmerhash** (int mode)
- void **link_kmerhash_to_dna_sequence** (kmerhash kmer, char ∗dna, size_t dna_length)
- void **del_kmerhash** (kmerhash kmer)
- bool **kmerhash_iterator** (kmerhash kmer)

Variables

- const char ∗ **biomcmc_kmer_class_string** [ ]

4.15.1 Detailed Description

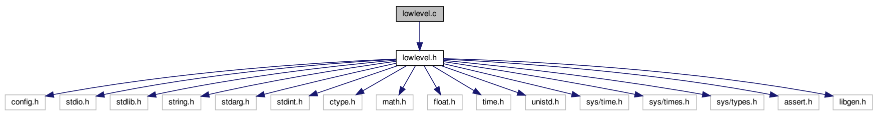
k-mer handling of DNA sequences, with hash transformation

4.16 lowlevel.c File Reference

Lowest level basic functions, that should be available to all other modules.

```
#include "lowlevel.h"
```

Include dependency graph for lowlevel.c:



Macros

- #define **MIN_CHUNK** 128

Functions

- void **hungarian_solve_integer** (hungarian p, int this_size)
- void **hungarian_solve_double** (hungarian p, int this_size)
- void ∗ biomcmc_malloc (size_t size)

  Memory-safe malloc() function.
- void ∗ biomcmc_realloc (void ∗ptr, size_t size)

  Memory-safe realloc() function.
- FILE ∗ biomcmc_fopen (const char ∗path, const char ∗mode)

  Memory-safe fopen() function.
- void biomcmc_error (const char ∗template,...)

  Prints error message and quits program.
- int compare_int_increasing (const void ∗a, const void ∗b)

  Comparison between integers, doubles, etc. used by qsort()

- int **compare_int_decreasing** (const void ∗a, const void ∗b)
- int **compare_uint64_increasing** (const void ∗a, const void ∗b)
- int **compare_uint64_decreasing** (const void ∗a, const void ∗b)
- int **compare_double_increasing** (const void ∗a, const void ∗b)
- int **compare_double_decreasing** (const void ∗a, const void ∗b)
- int biomcmc_getline (char ∗∗lineptr, size_t ∗n, FILE ∗stream)

  read file line-by-line (like homonymous function from GNU C library)
- uint32_t biomcmc_levenshtein_distance (const char ∗s1, uint32_t n1, const char ∗s2, uint32_t n2, uint32_t cost_sub, uint32_t cost_indel, bool skip_borders)

  edit distance between two sequences (slow), with option to allow one of sequences to terminate soon (o.w. global cost from end to end)
- void **hungarian_reset** (hungarian p)
- hungarian **new_hungarian** (int size, bool is_double)
- void **hungarian_update_cost** (hungarian p, int row, int col, void ∗cost)
- void **del_hungarian** (hungarian p)
- void **hungarian_solve** (hungarian p, int this_size)

4.16.1 Detailed Description

Lowest level basic functions, that should be available to all other modules.

4.16.2 Function Documentation

4.16.2.1 biomcmc_malloc()

```
void* biomcmc_malloc (
             size_t size )
```

Memory-safe malloc() function.

Allocates size bytes and returns a pointer to the allocated memory. An error message is thrown in case of failure.

Parameters

in	*size*	allocated size, in bytes

Returns

pointer to newly allocated memory

4.16.2.2 biomcmc_realloc()

```
void* biomcmc_realloc (
             void * ptr,
             size_t size )
```

Memory-safe realloc() function.

Changes the size of the memory block pointed to by ptr to size bytes. An error message is thrown in case of failure.

Parameters

in	*size*	allocated size, in bytes
in,out	*ptr*	pointer to previously allocated memory

Returns

pointer to newly allocated memory

4.16.2.3 biomcmc_fopen()

```
FILE* biomcmc_fopen (
             const char * path,
             const char * mode )
```

Memory-safe fopen() function.

Opens the file whose name is the string pointed to by path and associates a stream with it. An error message is thrown in case of failure.

Parameters

in	*path*	file name
in	*mode*	opening mode ("r" for reading, "w" for writing, etc)

Returns

pointer to file stream

4.16.2.4 biomcmc_error()

```
void biomcmc_error (
             const char * template,
              ... )
```

Prints error message and quits program.

similar to fprintf (stderr, ...), but exits after printing the message

Parameters

in	*template*	va_list following same format as printf()

Returns

exits program

4.16.2.5 biomcmc_getline()

```
int biomcmc_getline (
            char ** lineptr,
            size_t * n,
            FILE * stream )
```

read file line-by-line (like homonymous function from GNU C library)

This implementation is originally from the CvsGui project (`http://www.wincvs.org/`). The explanation from the original file adapted to our system follows:

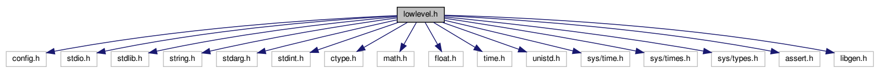
```
Read up to (and including) a newline ("\n") from STREAM into *LINEPTR and null-terminate it. *LINEPTR is a poi
returned from malloc (or NULL), pointing to *N characters of space.  It is realloc'd as necessary.  Return the
number of characters read (not including the null terminator), or -1 on error or EOF.
```

4.17 lowlevel.h File Reference

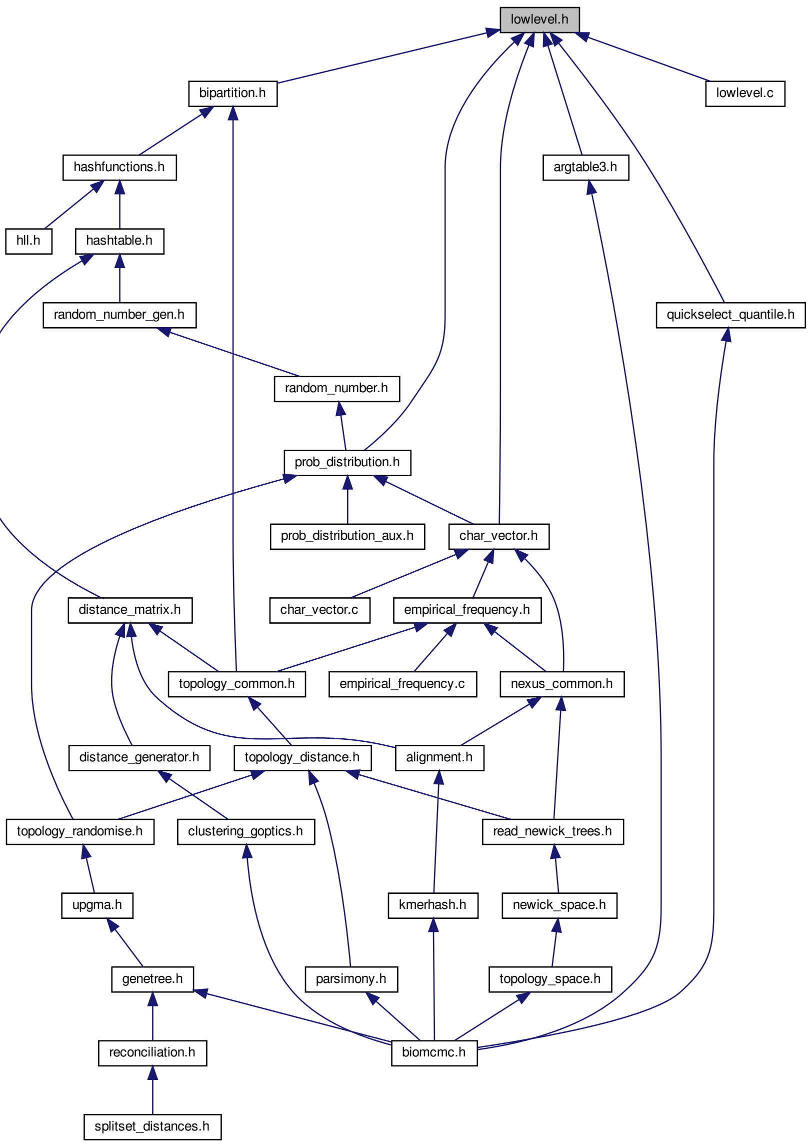
Lowest level header file. Header file for lowlevel.c.

```
#include "config.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#include <stdint.h>
#include <ctype.h>
#include <math.h>
#include <float.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/times.h>
#include <sys/types.h>
#include <assert.h>
#include <libgen.h>
```

Include dependency graph for lowlevel.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct hungarian_struct

Macros

- #define **EXP_1** 2.718281828459045235360287471352662 /∗ Euler's number ∗/
- #define true 1U
- #define false 0U
- #define **attribute_FALLTHROUGH** ((void)0);
- #define **MIN**(x, y) (((x)<(y)) ? (x) : (y))
- #define **MAX**(x, y) (((x)>(y)) ? (x) : (y))
- #define **MOD**(a) (((a)>0) ? (a) :(-a))

Typedefs

- typedef unsigned char bool

  Mnemonic for boolean (char is smaller than int)
- typedef struct hungarian_struct ∗ **hungarian**

Functions

- void ∗ biomcmc_malloc (size_t size)

  Memory-safe malloc() function.
- void ∗ biomcmc_realloc (void ∗ptr, size_t size)

  Memory-safe realloc() function.
- FILE ∗ biomcmc_fopen (const char ∗path, const char ∗mode)

  Memory-safe fopen() function.
- void biomcmc_error (const char ∗template,...)

  Prints error message and quits program.
- int compare_int_increasing (const void ∗a, const void ∗b)

  Comparison between integers, doubles, etc. used by qsort()
- int **compare_int_decreasing** (const void ∗a, const void ∗b)
- int **compare_uint64_increasing** (const void ∗a, const void ∗b)
- int **compare_uint64_decreasing** (const void ∗a, const void ∗b)
- int **compare_double_increasing** (const void ∗a, const void ∗b)
- int **compare_double_decreasing** (const void ∗a, const void ∗b)
- int biomcmc_getline (char ∗∗lineptr, size_t ∗n, FILE ∗stream)

  read file line-by-line (like homonymous function from GNU C library)
- uint32_t biomcmc_levenshtein_distance (const char ∗s1, uint32_t n1, const char ∗s2, uint32_t n2, uint32_t cost_sub, uint32_t cost_indel, bool skip_borders)

  edit distance between two sequences (slow), with option to allow one of sequences to terminate soon (o.w. global cost from end to end)
- hungarian **new_hungarian** (int size, bool is_double)
- void **hungarian_reset** (hungarian p)
- void **hungarian_update_cost** (hungarian p, int row, int col, void ∗cost)
- void **del_hungarian** (hungarian p)
- void **hungarian_solve** (hungarian p, int this_size)

4.17.1 Detailed Description

Lowest level header file. Header file for lowlevel.c.

4.17.2 Macro Definition Documentation

4.17.2.1 true

```
#define true 1U
```

Boolean TRUE

4.17.2.2 false

```
#define false 0U
```

Boolean FALSE

4.17.3 Function Documentation

4.17.3.1 biomcmc_malloc()

```
void* biomcmc_malloc (
            size_t size )
```

Memory-safe malloc() function.

Allocates size bytes and returns a pointer to the allocated memory. An error message is thrown in case of failure.

Parameters

in	*size*	allocated size, in bytes

Returns

pointer to newly allocated memory

4.17.3.2 biomcmc_realloc()

```
void* biomcmc_realloc (
            void * ptr,
            size_t size )
```

Memory-safe realloc() function.

Changes the size of the memory block pointed to by ptr to size bytes. An error message is thrown in case of failure.

Parameters

in	*size*	allocated size, in bytes
in,out	*ptr*	pointer to previously allocated memory

Returns

pointer to newly allocated memory

4.17.3.3 biomcmc_fopen()

```
FILE* biomcmc_fopen (
             const char * path,
             const char * mode )
```

Memory-safe fopen() function.

Opens the file whose name is the string pointed to by path and associates a stream with it. An error message is thrown in case of failure.

Parameters

in	*path*	file name
in	*mode*	opening mode ("r" for reading, "w" for writing, etc)

Returns

pointer to file stream

4.17.3.4 biomcmc_error()

```
void biomcmc_error (
             const char * template,
              ... )
```

Prints error message and quits program.

similar to fprintf (stderr, ...), but exits after printing the message

Parameters

in	*template*	va_list following same format as printf()

Returns

exits program

4.17.3.5 biomcmc_getline()

```
int biomcmc_getline (
             char ** lineptr,
             size_t * n,
             FILE * stream )
```

read file line-by-line (like homonymous function from GNU C library)

This implementation is originally from the CvsGui project (http://www.wincvs.org/). The explanation from the original file adapted to our system follows:

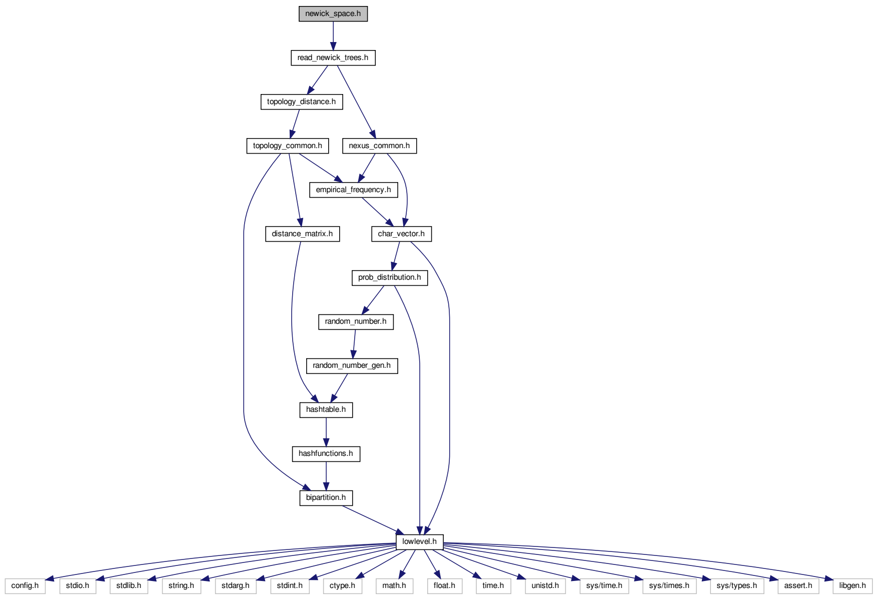
```
Read up to (and including) a newline ("\n") from STREAM into *LINEPTR and null-terminate it. *LINEPTR is a poi
returned from malloc (or NULL), pointing to *N characters of space.  It is realloc'd as necessary.  Return the
number of characters read (not including the null terminator), or -1 on error or EOF.
```

4.18 newick_space.h File Reference

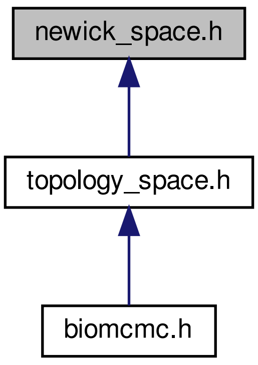
Reads a list of trees in newick format and creates vector of topologies.

```
#include "read_newick_trees.h"
```

Include dependency graph for newick_space.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct newick_space_struct

  Collection of topologies from tree file. Each topology will have its own char_vector.

Typedefs

- typedef struct newick_space_struct ∗ **newick_space**

Functions

- newick_space **new_newick_space** ()
- void **del_newick_space** (newick_space nwk)
- topology new_single_topology_from_newick_file (char ∗filename)

  Convenience function to read one newick tree from file, skipping checks (comments, multiline trees, etc.)
- newick_space **new_newick_space_from_file** (char ∗filename)
- void **update_newick_space_from_file** (newick_space nwk, char ∗filename)
- void **update_newick_space_from_string** (newick_space nwk, char ∗tree_string, size_t string_size)
- void **update_newick_space_from_topology** (newick_space nwk, topology topol)

4.18.1 Detailed Description

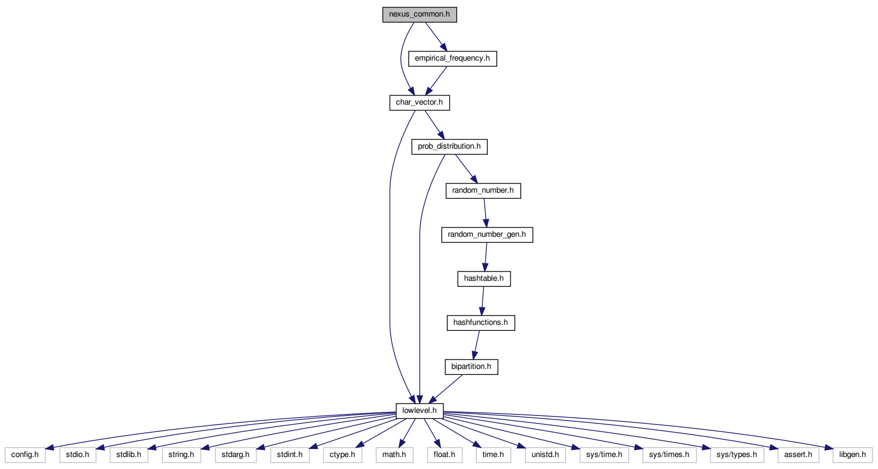
Reads a list of trees in newick format and creates vector of topologies.

Currently does not check for duplicated trees, or same leaf names on a tree

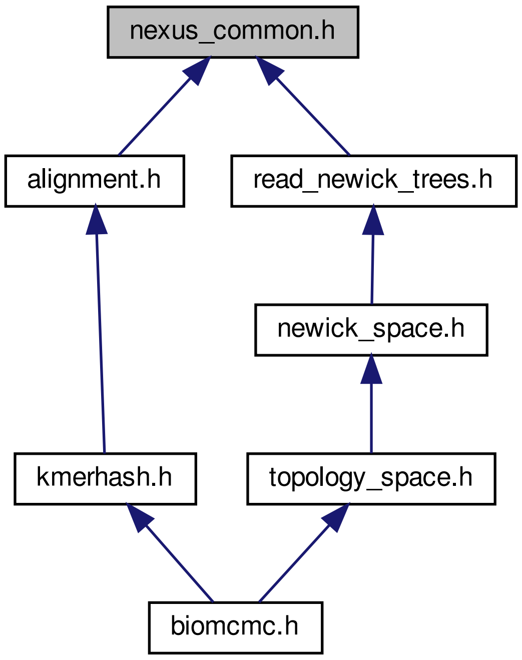
4.19 nexus_common.h File Reference

File handling functions for nexus format in general.

```
#include "char_vector.h"
#include "empirical_frequency.h"
```
Include dependency graph for nexus_common.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define MAX_NAME_LENGTH 4096

    maximum name length for taxa (alignment and tree files).

Functions

- char_vector new_char_vector_from_file (char ∗filename)

    General function that stores file content into char_vector_struct, removing shell-type comments.
- char ∗ remove_nexus_comments (char ∗∗string, size_t ∗stringsize, FILE ∗stream)

    Removes (possible nested/multiline) nexus comments of the form [] (brackets).
- char ∗ lowercase_string (char ∗string)

    Changes uppercase characters by lowercase versions.
- char ∗ uppercase_string (char ∗string)

    Changes lowercase characters by uppercase versions.
- char ∗ remove_space_from_string (char ∗string)

    Removes spaces, tabs from string.
- bool nonempty_string (char ∗string)

    returns bool::false if string is composed only of space characters (' ', '
    ', '', '', etc).
- bool nonempty_fasta_line (char ∗string)

    returns bool::false if first nonspace character of string is ';' (FASTA comment) or '#' (HUPO extension)

4.19.1 Detailed Description

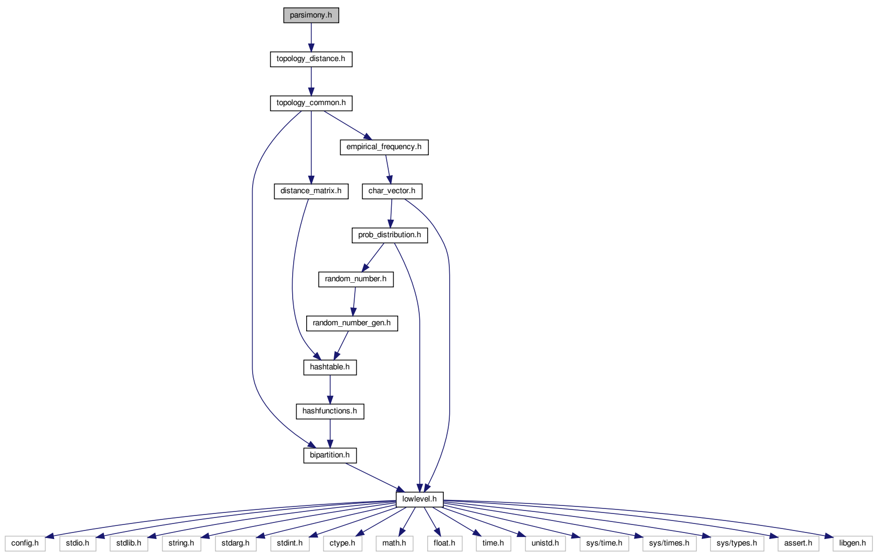
File handling functions for nexus format in general.

4.20 parsimony.h File Reference

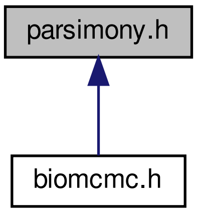
binary and multistate parsimony matrices, together with bipartition extraction for MRP

```
#include "topology_distance.h"
```

Include dependency graph for parsimony.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct binary_parsimony_datamatrix_struct

  used by matrix representation with parsimony (01 10 11 sequences)
- struct binary_parsimony_struct

Typedefs

- typedef struct binary_parsimony_datamatrix_struct ∗ **binary_parsimony_datamatrix**
- typedef struct binary_parsimony_struct ∗ **binary_parsimony**

Functions

- binary_parsimony_datamatrix **new_binary_parsimony_datamatrix** (int n_sequences)
- binary_parsimony_datamatrix **new_binary_parsimony_datamatrix_fixed_length** (int n_sequences, int n←
_sites)
- void **del_binary_parsimony_datamatrix** (binary_parsimony_datamatrix mrp)
- binary_parsimony **new_binary_parsimony** (int n_sequences)
- binary_parsimony **new_binary_parsimony_fixed_length** (int n_sequences, int n_sites)
- void **del_binary_parsimony** (binary_parsimony pars)
- void update_binary_parsimony_from_topology (binary_parsimony pars, topology t, int ∗map, int n_species)

  given a map[] with location in sptree of gene tree leaves, update binary matrix with splits from genetree
- int **binary_parsimony_score_of_topology** (binary_parsimony pars, topology t)
- void **pairwise_distances_from_binary_parsimony_datamatrix** (binary_parsimony_datamatrix mrp, double ∗∗dist, int n_dist)

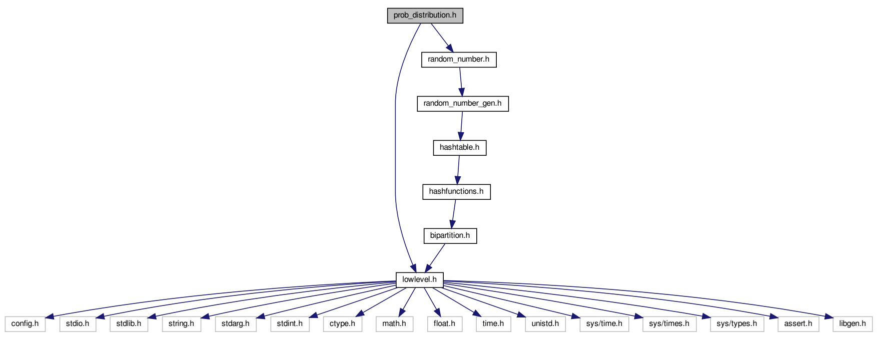
4.20.1 Detailed Description

binary and multistate parsimony matrices, together with bipartition extraction for MRP

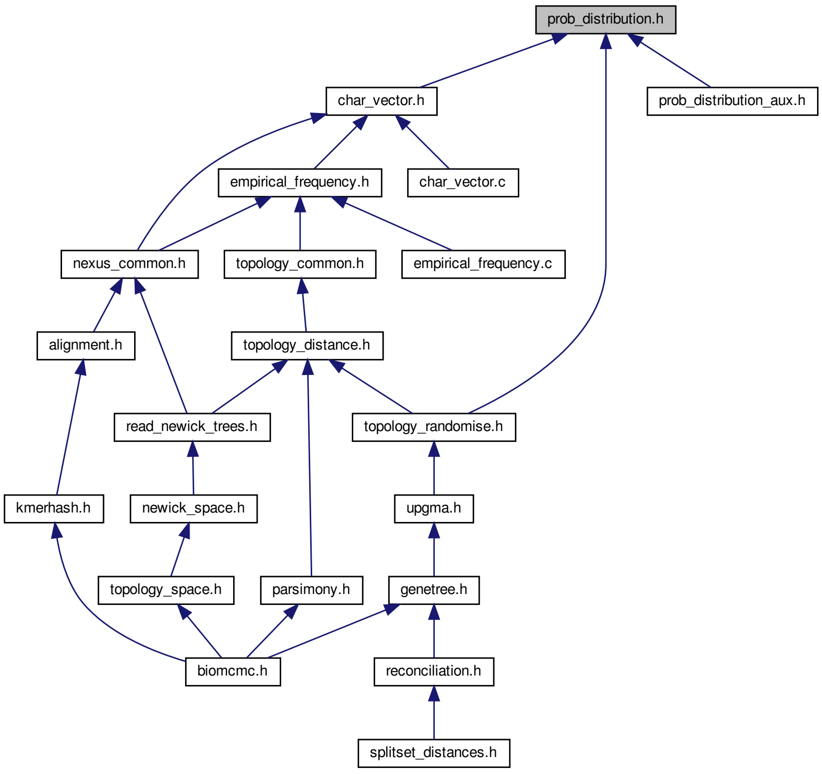
4.21 prob_distribution.h File Reference

Probability distribution functions and auxiliary mathematical functions from statistical package R.

```
#include "lowlevel.h"
#include "random_number.h"
```
Include dependency graph for prob_distribution.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct discrete_sample_struct

Typedefs

- typedef struct discrete_sample_struct ∗ **discrete_sample**

Functions

- void biomcmc_discrete_gamma (double alpha, double beta, double ∗rate, int nrates)

  Ziheng Yang's gamma discretization of rates.
- double biomcmc_dexp_dt (double d, double lambda, double m, bool log_p)

  pdf of discrete truncated exponential (d is discrete, m is maximum value)
- double biomcmc_pexp_dt (double d, double lambda, double m, bool log_p)

  cdf of discrete truncated exponential (d is discrete, m is maximum value): calculates P(D <= d)
- double biomcmc_qexp_dt (double p, double lambda, double m, bool log_p)

  quantile of discrete truncated exponential, that is, finds d s.t. P(D <= d) >= p
- double biomcmc_dgamma (double x, double alpha, double beta, bool log_p)

  gamma density
- double biomcmc_qgamma (double p, double alpha, double beta, bool log_p)

gamma quantile (inverse CDF)

- double biomcmc_pgamma (double x, double alpha, double beta, bool log_p)

    computes the cummulative distribution function for the gamma distribution with shape parameter alpha and rate parameter beta, s.t. $E[X] = \alpha/\beta$. The same as the (lower) incomplete gamma function.
- double **biomcmc_dnorm** (double x, double mu, double sigma, bool log_p)
- double **biomcmc_qnorm** (double p, double mu, double sigma, bool log_p)
- double **biomcmc_pnorm** (double x, double mu, double sigma, bool log_p)
- double **biomcmc_dlnorm** (double x, double meanlog, double sdlog, bool log_p)
- double **biomcmc_qlnorm** (double p, double meanlog, double sdlog, bool log_p)
- double **biomcmc_plnorm** (double x, double meanlog, double sdlog, bool log_p)
- double **biomcmc_dpois** (double x, double lambda, bool log_p)
- double **biomcmc_qpois** (double p, double lambda, bool log_p)
- double **biomcmc_ppois** (double x, double lambda, bool log_p)
- double **biomcmc_rng_gamma** (double alpha, double beta)
- double biomcmc_rng_norm (double mu, double sigma)

    Returns a random number from a Normal distribution N(mu, sigma^2) using 52 bits of precision.
- double **biomcmc_rng_lnorm** (double meanlog, double sdlog)
- double **biomcmc_rng_pois** (double mu)
- double **biomcmc_lgammafn** (double x, int ∗sgn)
- double **biomcmc_gammafn** (double x)
- double biomcmc_log1p (double x)

    compute the relative error logarithm $\log(1 + x)$ (C99 standard)
- double biomcmc_log1pmx (double x)

    accurate calculation of $\log(1 + x) - x$, particularly for small x
- double biomcmc_expm1 (double x)

    compute $\exp(x) - 1$ accurately also when x is close to zero, i.e. $|x| \ll 1$
- discrete_sample **new_discrete_sample_from_frequencies** (double ∗prob, size_t size)
- void **del_discrete_sample** (discrete_sample g)
- size_t **biomcmc_rng_discrete** (discrete_sample g)
- double **biomcmc_discrete_sample_pdf** (discrete_sample g, size_t k)
- double **biomcmc_logspace_add** (double logx, double logy)
- double **biomcmc_logspace_sub** (double logx, double logy)
- bool biomcmc_isfinite (double x)

    check if number is between minus infinity and plus infinity, or NaN

4.21.1 Detailed Description

Probability distribution functions and auxiliary mathematical functions from statistical package R.

Code derived from the `R project for Statistical Computing` version 2.9.1, available under the GPL license. It might be possible to use directly the standalone mathematical library "Rmath.h" from the R project instead of our implementation. The advantage would be a library updated more often than mine, at the cost of delegating to the guenomu user the installation and maintenance of the extra libraries (like GSL, for instance). In Debian this library can be installed through the package "r-mathlib". The original R library checks for several built-in compiler functions (like log1p(e) for calculating log(e+1)) but I simply assume the compiler has none and reimplement them. The CDFs always assume the lower tail (upper tails must use 1. - lower tail) or equivalent.

The code for the discrete sampling comes from the GNU Scientific Library version 1.14

4.21.2 Function Documentation

4.21.2.1 biomcmc_dexp_dt()

```
double biomcmc_dexp_dt (
            double d,
            double lambda,
            double m,
            bool log_p )
```

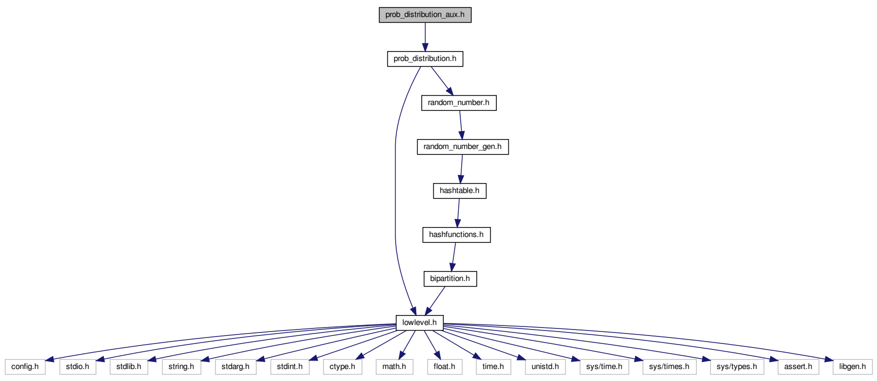
pdf of discrete truncated exponential (d is discrete, m is maximum value)

pdf of discrete truncated exponential (d is discrete, m is maximum value)

4.22 prob_distribution_aux.h File Reference

Auxiliary (low level) functions for prob_distribution.c.

`#include "prob_distribution.h"`
Include dependency graph for prob_distribution_aux.h:



Macros

- #define **R_LN2** 0.693147180559945309417232121458 /∗ ln(2) ∗/
- #define **R_PI** 3.141592653589793238462643383280 /∗ pi ∗/
- #define **R_2PI** 6.283185307179586476925286766559 /∗ 2∗pi ∗/
- #define **R_EXP_M1** 0.367879441171442321595523770161 /∗ exp(-1) = 1/e ∗/
- #define **R_SQRT_32** 5.656854249492380195206754896838 /∗ sqrt(32) ∗/
- #define **R_1_SQRT_2PI** 0.398942280401432677939946059934 /∗ 1/sqrt(2∗pi) ∗/
- #define **R_LN_SQRT_2PI** 0.918938533204672741780329736406 /∗ log(sqrt(2∗pi)) = log(2∗pi)/2 ∗/
- #define **R_LN_SQRT_PId2** 0.225791352644727432363097614947 /∗ log(sqrt(pi/2)) ∗/

Functions

- double **lgammacor** (double x)
- int **chebyshev_init** (const double ∗dos, int nos, double eta)
- double **chebyshev_eval** (double x, const double ∗a, const int n)
- void **gammalims** (double ∗xmin, double ∗xmax)
- double **logcf** (double x, double i, double d, double eps)
- double **lgamma1p** (double a)
- double **dpois_wrap** (double x_plus_1, double lambda, bool log_p)
- double **dpois_raw** (double x, double lambda, bool log_p)
- double **stirlerr** (double n)
- double **bd0** (double x, double np)
- double **pgamma_smallx** (double x, double alph, bool log_p)
- double **pd_upper_series** (double x, double y, bool log_p)
- double **pd_lower_series** (double lambda, double y)
- double **pd_lower_cf** (double i, double d)
- double **dpnorm** (double x, double lp)
- double **ppois_asymp** (double x, double lambda, bool log_p)
- double **pgamma_raw** (double x, double alph, bool log_p)
- double **qchisq_appr** (double p, double nu, double g, bool log_p, double tol)
- void **pnorm_both** (double x, double ∗cum, double ∗ccum, int i_tail, bool log_p)
- double **do_poisson_search** (double y, double ∗z, double p, double lambda, double incr)

Variables

- const double **pInf** = 1./0.
- const double **mInf** = -1./0.
- const double **NaN** = 0./0.
- const double **scalefactor** = 115792089237316195423570985008687907853269984665640564039457584007913129639936
- const double **M_cutoff** = R_LN2 ∗ DBL_MAX_EXP / DBL_EPSILON

4.22.1 Detailed Description

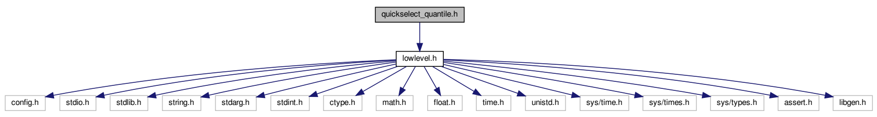
Auxiliary (low level) functions for prob_distribution.c.

4.23 quickselect_quantile.h File Reference

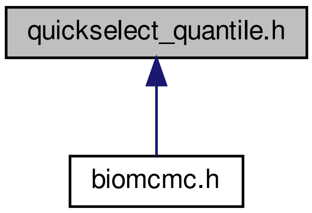
find k smallest element in vector

```
#include "lowlevel.h"
```

Include dependency graph for quickselect_quantile.h:



This graph shows which files directly or indirectly include this file:



Functions

- double **biomcmc_quantile_double** (double ∗original_vector, int n, double quantile)
- void **biomcmc_quantile_vector_double** (double ∗original_vector, int n, double ∗quantile, int n_quantile, double ∗result)
- double biomcmc_wirth_algorithm (double ∗a, int n, int k)

  find k-smallest element, changing vector a[]

4.23.1 Detailed Description

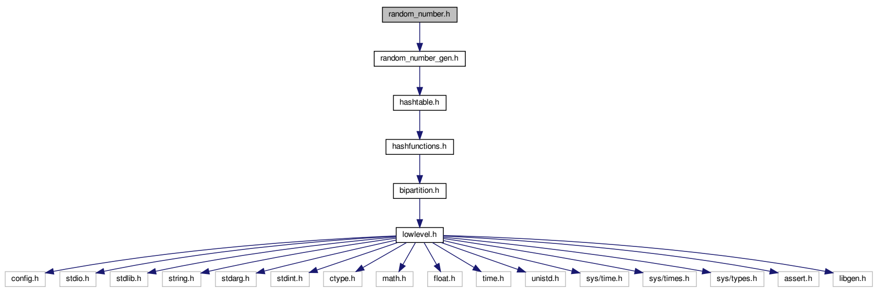
find k smallest element in vector

4.24 random_number.h File Reference

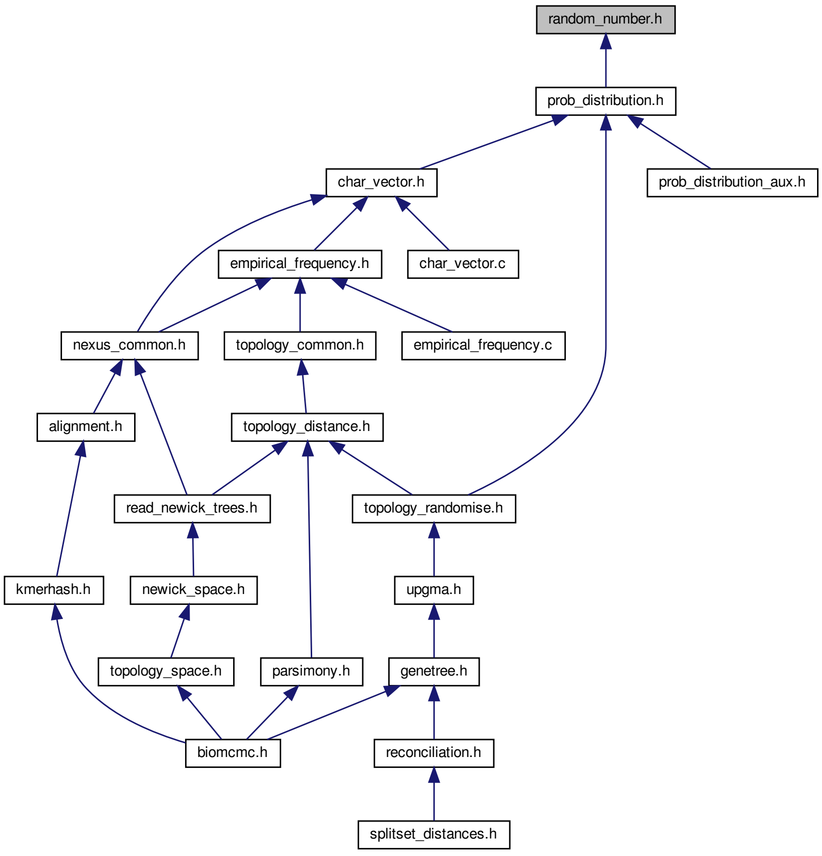
Random number generation, with algorithms from the Gnu Scientific Library (GSL) and motivation from the Scalable Parallel Pseudo Random Number Generators Library (SPRNG).

```
#include "random_number_gen.h"
```

Include dependency graph for random_number.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct biomcmc_rng_struct

  Random number structure (combined Tausworthe algorithm)

Typedefs

- typedef struct biomcmc_rng_struct $*$ **biomcmc_rng**

Functions

- void biomcmc_random_number_init (unsigned long long int seed)

  High-level setup of a simple random number generator and initialization with a seed (not to be mixed with other low-level functions that update the seed or allocate memory).
- void biomcmc_random_number_finalize (void)

  High-level finalization (memory release etc.) of the random number environment.
- biomcmc_rng new_biomcmc_rng (unsigned long long int seed, int stream_number)

  Allocate memory for new (Tausworthe + MT19937) generator from a pool of streams.
- void del_biomcmc_rng (biomcmc_rng r)

Release memory occupied by biomcmc_rng::

- unsigned long long int biomcmc_rng_get_initial_seed (void)

  Create initial seed based on time, combining time in microseconds and seconds (user-controled seed is not uint64_t)
- biomcmc_rng new_biomcmc_rng_from_seed (unsigned long long int seed, int stream_number)

  Generate a vector of seeds (based on initial one), create and initialize stream with an element of this vector.
- double biomcmc_rng_snorm32 (void)

  Returns a random number from a Standard Normal distribution N(0,1) - prob_distribution.h has general case.
- double biomcmc_rng_snorm (void)

  Returns a random number from a Standard Normal distribution with maximum (52 bits) integer precision.
- double biomcmc_rng_unif (void)

  Returns a random number between 0 and 1 (including 1) with precision $\approx 10^{-15}$ *(52 bits).*
- double biomcmc_rng_unif_pos (void)

  Returns a positive random number between 0 and 1 (excluding 0 and including 1) (52 bits).
- uint64_t biomcmc_rng_unif_int64 (uint64_t n)

  Returns a long integer (64 bits) random number between 0 and n (excluding n), with $n < 10^{15}$.
- double biomcmc_rng_unif32 (void)

  Returns a random number between 0 and 1 (including 1) with precision $\approx 10^{-9}$ *(32 int bits).*
- double biomcmc_rng_unif_pos32 (void)

  Returns a positive random number between 0 and 1 (including 1) (32 int bits).
- uint32_t biomcmc_rng_unif_int (uint32_t n)

  Returns an integer (32 bits) random number between 0 and n (excluding n), provided $n < 410^9$ *approx.*
- uint64_t biomcmc_rng_get (void)

  new value with 64 random bits
- double biomcmc_rng_get_52 (void)

  new value with 52 random bits as a double precision float
- uint32_t biomcmc_rng_get_32 (void)

  new value with 32 random bits
- void biomcmc_get_time (int $*$time)

  get current time with maximum precision and soter in vector time[2]
- double biomcmc_elapsed_time (int $*$now, int $*$past)

  returns the floating-point time in seconds elapsed between past[2] and now[2]

Variables

- biomcmc_rng biomcmc_random_number

  pointer to pseudo-random number generator (should point to real stream, even when there are several)

4.24.1 Detailed Description

Random number generation, with algorithms from the Gnu Scientific Library (GSL) and motivation from the Scalable Parallel Pseudo Random Number Generators Library (SPRNG).

There are two ways of setting up the pseudo-random number generation: a high-level approach where we simply call one function before start using the generator and another after we finished using it, and a lower-level approach where we must allocate, seed and free the streams by hand. Both approaches are completely incompatible, and there is no check to avoid this mistake so the programmer should use the first, high-level approach unless he understand well the algorithms. The high-level approach is useful when one stream is enough (for instance, a serial program). In this case we use a (modified) maximally-equidistributed combined Linear Feedback Shift Register (LFSR, or Tausworthe) pseudo-random number generator (PRNG) whose seed is based on time of day.

The low level approach is useful if one needs several uncorrelated streams - for example a parallel program where each node must have its own stream of pseudo-random numbers and a common stream shared by all nodes. For this we implemented a modified `random tree method` where a lagged-Fibonacci generator (two-tap Generalised Feedback Shift Register - GFSR4) PRNG is used to generate the seeds for the (modified) Tausworthe streams. The seed for this GFSR4 generator is given by the time of day and should be set only once by the program, given its low randomness. The GFSR4 uses a vector of size 2^14 which is initialized by an (quick-and-dirty) xorshift randomization of the seed, which makes it sensitive to the choice of this seed. I included a `gamerand fast randomization` over each element of the vector, modified for 64 bits (the algorithm is too simple to require a license and I've seen public domain versions of it; the original disclaimer is GPL-like). A more robust alternative **not implemented here** would be to use a better PRNG to feed the initial vector of the GFSR4 generator. As noticed by `CJK Tan and JAR Blais (HPCN 2000, LNCS 1823, 127-135)` another random tree method could be constructed with parallel GFSR4 streams where the initial states are created by an PRNG - equivalent to our seed generator, but providing not only the seed but all 2^{14} elements to each stream. To ensure independence of streams - since different seeds are just different points of the same stream - we implemented all 150 parameter sets provided by `L'ecuyer (Maths Comput 1999, pp.261)` for the Tausworthe generators. So we have at least 150 independent streams, with periods between 10^{14} and 10^{35} - if I understood correctly the interpretation of the number of non-zero solutions of the polynomials N_1.

Our modification to the original Tausworthe (LFSR) algorithms is to combine it with the `generalized Marsaglias's xorshift PRNG called xorgens`, developed by Richard Brent and available under a GPL license. So the LFSR has one extra component from this independent xorshift (combined through a XOR). The seed for the xorshift is the same as for the Tausworthe.

All streams are of 64 bits (dependent on a "long long int" having 64 bits) but 32 bits or even 16 bits are available through wrapper functions. If your system does not provide 64 bit integers use the unwrapped versions instead. When working with more than one stream at the same time - by the same node, using the example of a parallel program - we must update by hand the variable pointed to by the global variable ::random_number to indicate which stream should be used.

Some original comments from the GSL can be found on "doc/random_number_generation.txt"

4.24.2 Function Documentation

4.24.2.1 biomcmc_random_number_init()

```
void biomcmc_random_number_init (
            unsigned long long int seed )
```

High-level setup of a simple random number generator and initialization with a seed (not to be mixed with other low-level functions that update the seed or allocate memory).

The seed may be provided by calling function (mostly for debug) if not zero otherwise it is based on present time of day, and uses the Tausworthe pseudo-random number generator. The Tausworthe generator we use has a period of (at least?) 10^{35}. This function allocates memory to global variable ::random_number directly

4.25 random_number_aux.h File Reference

Variables and structures local to random_number.c (should be opaque to user)

Variables

- uint64_t sTable76 [44][5]

  Five-element streams for L'ecuyer's combined LFSR (Tausworthe) generator.
- uint64_t sTable543 [106][4]

  Four-element streams for L'ecuyer's combined LFSR (Tausworthe) generator.
- uint64_t **qTable76** [2][5] = { {1ULL, 7ULL, 24ULL, 3ULL, 5ULL}, {1ULL, 24ULL, 3ULL, 5ULL, 3ULL} }
- uint64_t **kTable76** [2][5] = { {63ULL, 57ULL, 55ULL, 52ULL, 47ULL}, {63ULL, 55ULL, 52ULL, 47ULL, 41ULL} }
- uint64_t **qTable543** [4][4]
- uint64_t **kTable543** [4][4]
- uint64_t **Cmask** [28]
- uint32_t **marsaglia_constants** [81]

4.25.1 Detailed Description

Variables and structures local to random_number.c (should be opaque to user)

4.25.2 Variable Documentation

4.25.2.1 sTable76

```
uint64_t sTable76[44][5]
```

Initial value:

```
= {

   {9ULL, 34ULL, 5ULL, 26ULL, 18ULL},  {9ULL, 32ULL, 5ULL, 31ULL, 6ULL},   {9ULL, 25ULL, 5ULL, 37ULL, 22ULL
      },
   {10ULL, 24ULL, 5ULL, 7ULL, 12ULL},  {12ULL, 17ULL, 5ULL, 14ULL, 8ULL},  {12ULL, 40ULL, 5ULL, 16ULL, 22
      ULL},
   {12ULL, 26ULL, 5ULL, 34ULL, 23ULL}, {17ULL, 27ULL, 5ULL, 13ULL, 9ULL},  {17ULL, 8ULL, 5ULL, 37ULL, 19ULL
      },
   {20ULL, 41ULL, 5ULL, 14ULL, 6ULL},  {22ULL, 40ULL, 5ULL, 4ULL, 18ULL},  {22ULL, 19ULL, 5ULL, 14ULL, 19
      ULL},
   {22ULL, 41ULL, 5ULL, 16ULL, 6ULL},  {22ULL, 16ULL, 5ULL, 32ULL, 4ULL},  {26ULL, 9ULL, 5ULL, 11ULL, 14ULL
      },
   {26ULL, 19ULL, 5ULL, 29ULL, 3ULL},  {44ULL, 20ULL, 5ULL, 8ULL, 6ULL},   {44ULL, 31ULL, 5ULL, 22ULL, 14
      ULL},
   {53ULL, 8ULL, 5ULL, 23ULL, 17ULL},  {53ULL, 12ULL, 5ULL, 31ULL, 18ULL},

   {10ULL, 5ULL, 29ULL, 23ULL, 8ULL},  {12ULL, 5ULL, 11ULL, 16ULL, 15ULL}, {17ULL, 5ULL, 16ULL, 6ULL, 7ULL}
      ,
   {17ULL, 5ULL, 19ULL, 16ULL, 14ULL}, {18ULL, 5ULL, 37ULL, 7ULL, 3ULL},   {19ULL, 5ULL, 31ULL, 15ULL, 13
      ULL},
   {20ULL, 5ULL, 11ULL, 13ULL, 6ULL},  {22ULL, 5ULL, 17ULL, 10ULL, 11ULL}, {23ULL, 5ULL, 37ULL, 13ULL, 7ULL
      },
   {24ULL, 5ULL, 7ULL, 16ULL, 8ULL},   {26ULL, 5ULL, 22ULL, 4ULL, 9ULL},   {26ULL, 5ULL, 26ULL, 13ULL, 12
      ULL},
   {26ULL, 5ULL, 31ULL, 14ULL, 13ULL}, {36ULL, 5ULL, 32ULL, 16ULL, 8ULL},  {36ULL, 5ULL, 32ULL, 21ULL, 8ULL
      },
   {39ULL, 5ULL, 19ULL, 6ULL, 8ULL},   {43ULL, 5ULL, 14ULL, 20ULL, 15ULL}, {44ULL, 5ULL, 14ULL, 15ULL, 15
      ULL},
   {44ULL, 5ULL, 29ULL, 6ULL, 13ULL},  {44ULL, 5ULL, 34ULL, 25ULL, 9ULL},  {45ULL, 5ULL, 16ULL, 21ULL, 8ULL
      },
   {51ULL, 5ULL, 28ULL, 3ULL, 12ULL},  {53ULL, 5ULL, 26ULL, 16ULL, 8ULL},  {54ULL, 5ULL, 28ULL, 13ULL, 3ULL
      }
}
```

Five-element streams for L'ecuyer's combined LFSR (Tausworthe) generator.

4.25.2.2 qTable543

```
uint64_t qTable543[4][4]
```

Initial value:

```
= {
   {31ULL, 1ULL, 19ULL, 22ULL}, {31ULL, 11ULL, 19ULL, 22ULL}, {1ULL, 19ULL, 7ULL, 24ULL}, {31ULL, 19ULL, 24
      ULL, 21ULL}
}
```

4.25.2.3 kTable543

```
uint64_t kTable543[4][4]
```

Initial value:

```
= {
   {63ULL, 60ULL, 58ULL, 57ULL}, {63ULL, 60ULL, 58ULL, 57ULL}, {63ULL, 58ULL, 57ULL, 55ULL}, {63ULL, 58ULL,
       55ULL, 47ULL}
}
```

4.25.2.4 Cmask

```
uint64_t Cmask[28]
```

Initial value:

```
= {
  0xfffffffff0000000ULL, 0xfffffffff8000000ULL, 0xfffffffffc000000ULL, 0xfffffffffe000000ULL,
  0xffffffffff000000ULL, 0xffffffffff800000ULL, 0xffffffffffc00000ULL, 0xffffffffffe00000ULL,
  0xfffffffffff00000ULL, 0xfffffffffff80000ULL, 0xfffffffffffc0000ULL, 0xfffffffffffe0000ULL,
  0xffffffffffff0000ULL, 0xffffffffffff8000ULL, 0xffffffffffffc000ULL, 0xffffffffffffe000ULL,
  0xfffffffffffff000ULL, 0xfffffffffffff800ULL, 0xfffffffffffffc00ULL, 0xfffffffffffffe00ULL,
  0xffffffffffffff00ULL, 0xffffffffffffff80ULL, 0xffffffffffffffc0ULL, 0xffffffffffffffe0ULL,
  0xfffffffffffffff0ULL, 0xfffffffffffffff8ULL, 0xfffffffffffffffcULL, 0xfffffffffffffffeULL
}
```

4.25.2.5 marsaglia_constants

```
uint32_t marsaglia_constants[81]
```

Initial value:

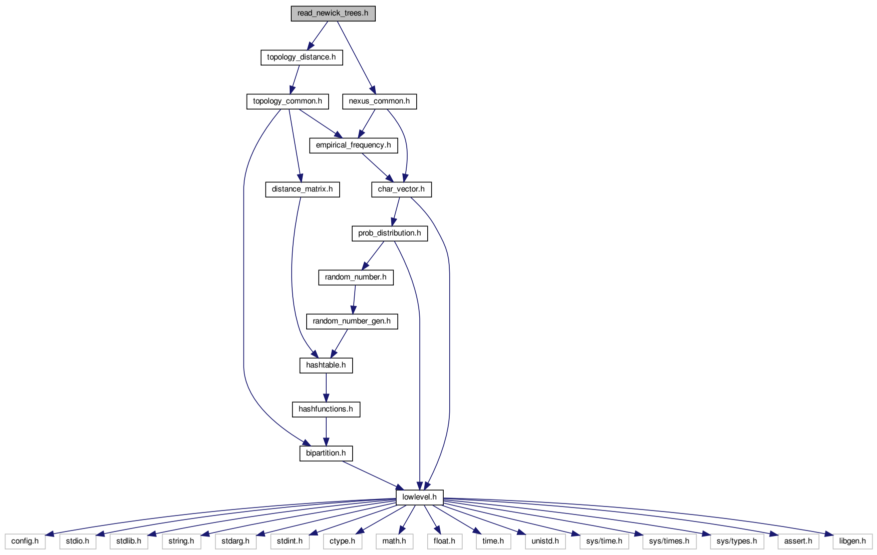
```
= {
  18000, 18030, 18273, 18513, 18879, 19074, 19098, 19164, 19215, 19584, 19599, 19950, 20088, 20508, 20544,
      20664, 20814,
  20970, 21153, 21243, 21423, 21723, 21954, 22125, 22188, 22293, 22860, 22938, 22965, 22974, 23109, 23124,
      23163, 23208,
  23508, 23520, 23553, 23658, 23865, 24114, 24219, 24660, 24699, 24864, 24948, 25023, 25308, 25443, 26004,
      26088, 26154,
  26550, 26679, 26838, 27183, 27258, 27753, 27795, 27810, 27834, 27960, 28320, 28380, 28689, 28710, 28794,
      28854, 28959,
  28980, 29013, 29379, 29889, 30135, 30345, 30459, 30714, 30903, 30963, 31059, 31083, 36969
}
```

4.26 read_newick_trees.h File Reference

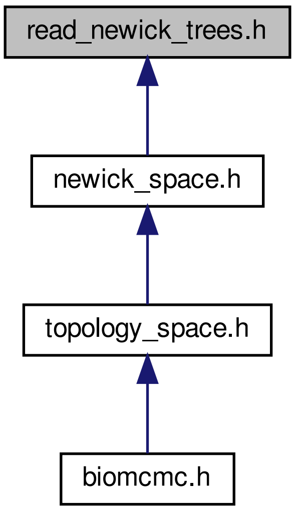
Low-level functions for reading newick strings.

```
#include "topology_distance.h"
#include "nexus_common.h"
```

Include dependency graph for read_newick_trees.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct newick_node_struct

    newick trees have minimal information, unlike topology_struct
- struct newick_tree_struct

Typedefs

- typedef struct newick_node_struct ∗ **newick_node**
- typedef struct newick_tree_struct ∗ **newick_tree**

Functions

- newick_tree new_newick_tree (int nleaves)

    Allocates memory for newick_tree_struct.
- void del_newick_tree (newick_tree T)

    Frees memory used by tree.
- void copy_topology_from_newick_tree (topology tree, newick_tree nwk_tree, bool create_tree_taxlabel)

    Copy information from newick_tree struct to topology_struct; newick_space copies taxlabels but topology_space (from nexus files) share the taxlabel and thus don't copy from newick_tree_struct.
- newick_tree new_newick_tree_from_string (char ∗external_string)

    Creates newick_tree structure.
- newick_node subtree_newick_tree (newick_tree tree, char ∗lsptr, char ∗rsptr, int ∗node_id, newick_node up)

    Recursive function that creates a node based on parenthetic structure.
- int number_of_leaves_in_newick (char ∗∗string, int ∗number_branches)

    Counts the number of leaves and resolves (one) trifurcation of tree string.

4.26.1 Detailed Description

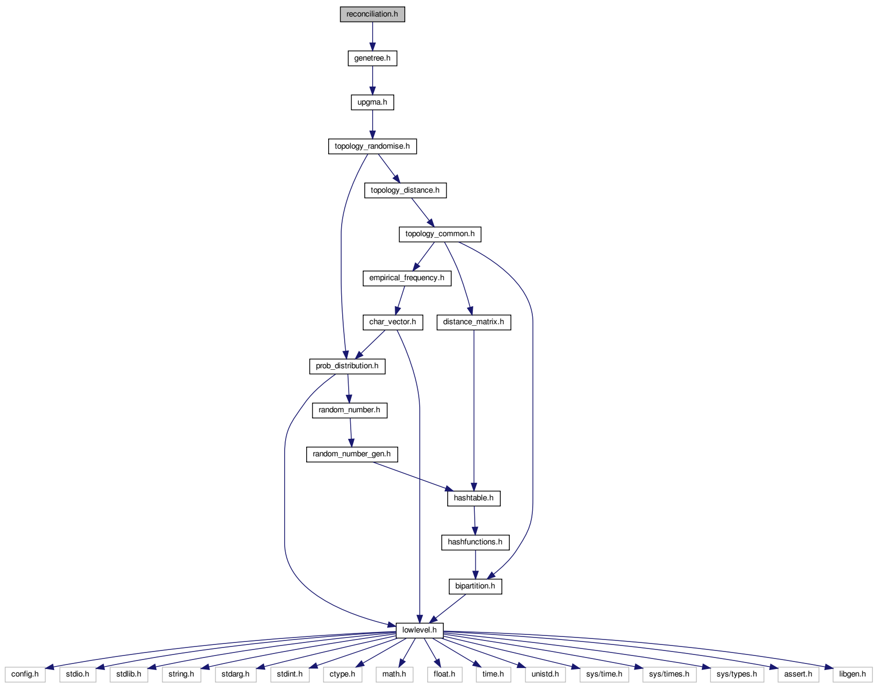
Low-level functions for reading newick strings.

Currently does not check for duplicated trees, or same leaf names on a tree

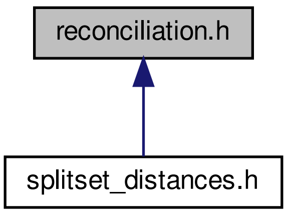
4.27 reconciliation.h File Reference

low-level file for gene tree and species tree reconciliation. This file is hidden from user and contains the LCA-based reconciliation distances.

`#include "genetree.h"`
Include dependency graph for reconciliation.h:



This graph shows which files directly or indirectly include this file:



Functions

- reconciliation new_reconciliation (int gene_nleaves, int sp_nleaves)

  Allocate space for new reconciliation_struct (other functions defined in topology_mrca.c)
- reconciliation new_reconciliation_from_reconciliation (int gene_nleaves, int sp_nleaves, reconciliation from)

  Create new reconciliation struct and copy values (except species tree info) from another struct.
- void del_reconciliation (reconciliation r)

  release allocated memory for reconciliation_struct
- void initialize_reconciliation_sp_count (reconciliation rec, int n_sp, int n_idx)

  Fill rec->sp_count[] with the number of representatives of each species (idexed by rec->sp_id[])

- void initialize_reconciliation_from_new_species_tree (genetree gtre, speciestree sptre)

  transform indexes found in index_sptaxa_to_genetaxa() to pointers to species nodes
- void reconciliation_gene_tree_reconcile (genetree gtre, speciestree sptre)

  Find reconciliation map between gene and species trees.

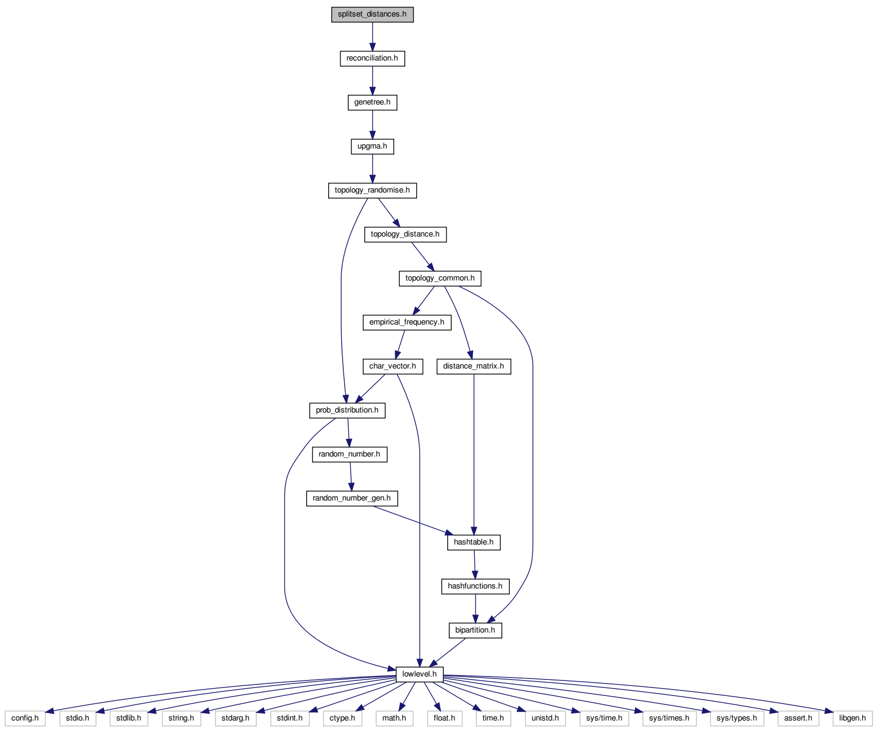
4.27.1 Detailed Description

low-level file for gene tree and species tree reconciliation. This file is hidden from user and contains the LCA-based reconciliation distances.

4.28 splitset_distances.h File Reference

Low-level functions that use only the split bipartitions of topologies – treating them as unrooted usually.

`#include "reconciliation.h"`
Include dependency graph for splitset_distances.h:



Functions

- splitset new_splitset_genespecies (topology gene, topology species, reconciliation rec)

  Splitset structure for dSPR calculation (also allocates aux vectors)
- void del_splitset (splitset split)

  free memory allocated for splitset structure
- int dSPR_gene_species (topology gene, topology species, splitset split)

  approximate dSPR between unrooted gene and species trees (leafset mapping)
- int dSPR_gene_species_rf (topology gene, topology species, splitset split)

  RF distance between unrooted gene and species trees (leafset mapping)
- int dSPR_gene_species_hdist (topology gene, topology species, splitset split)

  h distance (edge disagreement assignment cost) between unrooted gene and species trees (leafset mapping)

4.28.1 Detailed Description

Low-level functions that use only the split bipartitions of topologies – treating them as unrooted usually.

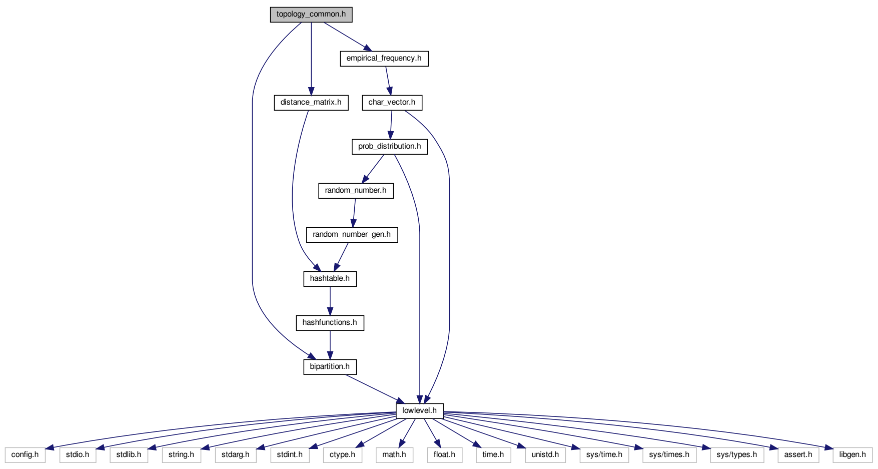
Use a "splitset" structure that copies the bipartition information of all nodes (so that original trees are untouched) and then modifies this splitset. These functions assume a gene tree (mul-tree) and a species tree (NOT mul-tree). Compared to guenomu and genefam-dist, I removed the simpler 'orthologous' functions since they assumed _↩
same_leaves on both trees, which is not usual even without multrees.

4.29 topology_common.h File Reference

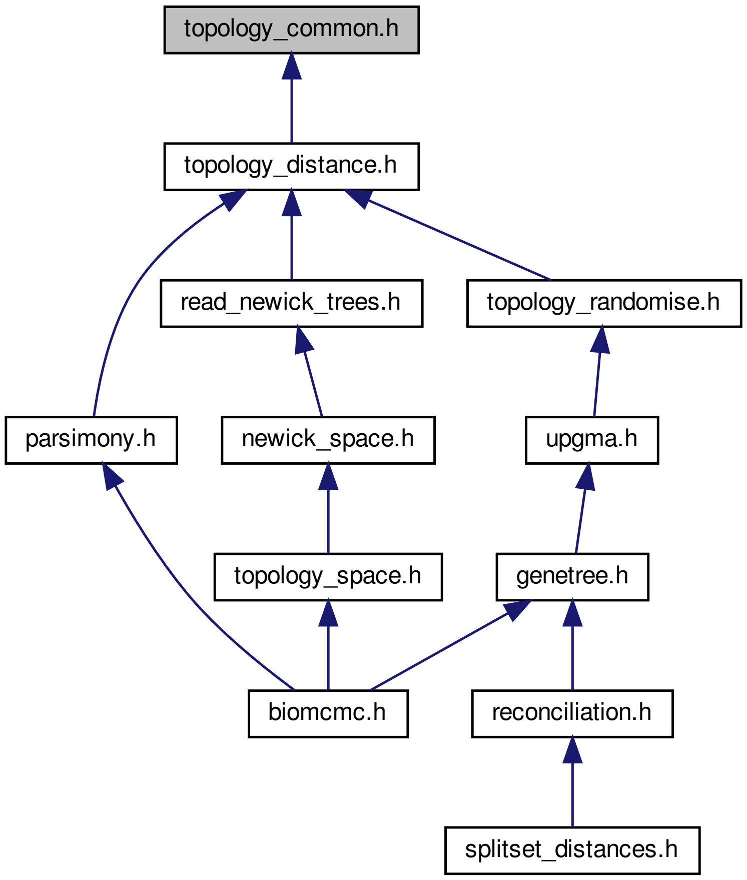
General-purpose topology structures created from nexus_tree_struct (and low-level functions)

```
#include "bipartition.h"
#include "distance_matrix.h"
#include "empirical_frequency.h"
```

Include dependency graph for topology_common.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct topol_node_struct

  Information of a node (binary tree).
- struct topology_struct

  Binary unrooted topology (rooted at leaf with ID zero)

Typedefs

- typedef struct topol_node_struct ∗ **topol_node**
- typedef struct topology_struct ∗ **topology**

Functions

- topology new_topology (int nleaves)

  Allocate space for new topology_struct.

- void topology_malloc_blength (topology tree)

  Allocate vector for branch lengths (3 vectors: mean, min and max values observed in topol_space collection)
- void del_topology (topology topol)

  Free space allocated by topology_struct.
- void **debug_topol** (topology tree)
- void copy_topology_from_topology (topology to_tree, topology from_tree)

  Copy information from topology_struct.
- void update_topology_sisters (topology tree)

  Update pointers to topol_node_struct::sister.
- void update_topology_traversal (topology tree)

  Update topol_node::preorder, topol_node::postorder, topol_node::bipartition and order siblings by number of descendants.
- bool topology_is_equal (topology t1, topology t2)

  Compare two topologies based on bipartitions as clades (not on branch lengths)
- bool topology_is_equal_unrooted (topology t1, topology t2, bool use_root_later)

  Compare two topologies based on bipartitions neglecting root; boolean ask if split should be reverted to original orientation.
- void reorder_topology_leaves (topology tree)

  Reorder char_vector_struct; leaf node ids (and bipartitions) must then follow this order.
- bool node1_is_child_of_node2 (topol_node node1, topol_node node2)

  Boolean if node2 is on the path of node1 to the root.
- char ∗ topology_to_string_by_id (const topology tree, double ∗blen)

  Print subtree in newick format to string using leaf IDs.
- char ∗ topology_to_string_create_name (const topology tree, double ∗blen)

  Print subtree in newick format to string creating names (based on leaf IDs.)
- char ∗ topology_to_string_by_name (const topology tree, double ∗blen)

  Print subtree in newick format to string using leaf names.
- void graphviz_file_topology (FILE ∗fout, char ∗label, const topology tree)

  Prints subtree in dot format to file.
- void apply_spr_at_nodes (topology p, topol_node prune, topol_node regraft, bool update_done)

  Apply one subtree prune-and-regraft (SPR branch swapping) operation at specified nodes.
- void apply_spr_at_nodes_LCAprune (topology tree, topol_node prune, topol_node regraft, bool update_↩ done)

  Apply one SPR branch swapping at specified nodes when prune subtree is above prune node.
- void apply_spr_at_nodes_notLCAprune (topology tree, topol_node prune, topol_node regraft, bool update↩ _done)

  Apply one SPR branch swapping at specified nodes when subtree to be pruned is below prune node.
- void topology_undo_random_move (topology tree, bool update_done)

  revert last SPR branch swapping
- void clear_topology_flags (topology tree)

  reset all d_done and u_done booleans to "true" (when rejecting a new state in MCMC)
- void raise_topology_flags (topology tree)

  reset all d_done and u_done booleans to "false" (when updating a model parameter with MTM)
- void topology_reset_random_move (topology tree)

  revert last SPR branch swapping and clear flags (reject last proposal, in MCMC)
- int copy_topology_to_intvector_by_postorder (topology tree, int ∗ivec)

  store ID of each node's parent (in postorder) into vector, returning number of stored nodes
- int copy_intvector_to_topology_by_postorder (topology tree, int ∗ivec)

  restore topological structure based on postordered ID vector, returning number of restored nodes
- void copy_topology_to_intvector_by_id (topology tree, int ∗ivec)

  store ID of each node's parent into vector
- void copy_intvector_to_topology_by_id (topology tree, int ∗ivec)

  restore topological structure based on ID vector

4.29.1 Detailed Description

General-purpose topology structures created from nexus_tree_struct (and low-level functions)

The topology structure should actually be called "tree" since it has information about branch lengths, but these functions neglect branch lenght information. Here we have functions that create split bipartitions for edges (stored by nodes below the edge) and compare distinct topologies based on these bipartitions. We also have here the lowest-level function that apply an SPR on a topology (again, without caring about the branch length).

4.29.2 Function Documentation

4.29.2.1 copy_topology_from_topology()

```
void copy_topology_from_topology (
            topology to_tree,
            topology from_tree )
```

Copy information from topology_struct.

Since IDs do not change, this function only needs to update topol_node_struct::up, topol_node_struct::right, and topol_node_struct::left pointers and topol_node_struct::map_id from internal nodes; update of topol_node::sister is handled by function update_topology_sisters().

Parameters

in	*from_tree*	original topology_struct
out	*to_tree*	(previously allocated) copied topology_struct

4.29.2.2 topology_to_string_by_id()

```
char* topology_to_string_by_id (
            const topology tree,
            double * blen )
```

Print subtree in newick format to string using leaf IDs.

Stores in string the tree in newick format, using leaf ID numbers (in practical applications needs a TRANSLATION nexus block). Memory allocation is handled by this function, but needs to be freed by the calling function.

Parameters

in	*tree*	tree to be printed
in	*blen*	vector with branch lengths (usually tree->blength)

Returns

a pointer to newly allocated string

4.29.2.3 topology_to_string_create_name()

```
char* topology_to_string_create_name (
            const topology tree,
            double * blen )
```

Print subtree in newick format to string creating names (based on leaf IDs.)

Stores in string the tree in newick format, using newly-created names based on leaf ID numbers (useful for generating random trees that must be read by other programs.) Memory allocation is handled by this function, but needs to be freed by the calling function.

Parameters

in	*tree*	tree to be printed
in	*blen*	vector with branch lengths (usually tree->blength)

Returns

a pointer to newly allocated string

4.29.2.4 topology_to_string_by_name()

```
char* topology_to_string_by_name (
            const topology tree,
            double * blen )
```

Print subtree in newick format to string using leaf names.

Stores in string the tree in newick format, preserving sequence names if available. Memory allocation is handled by this function, but needs to be freed by the calling function.

Parameters

in	*tree*	tree to be printed
in	*blen*	vector with branch lengths (usually tree->blength)

Returns

a pointer to newly allocated string

4.29.2.5 graphviz_file_topology()

```
void graphviz_file_topology (
            FILE * fout,
            char * label,
            const topology tree )
```

Prints subtree in dot format to file.

Prints to file the tree in dot format (undirected graph). The dot format can be used with the `graphviz` suite of programs, and is not restricted to trees but can also handle arbitrary graph structures. Notice that we do not make use of the graphviz library, we simply create the text file graphviz programs take as input. Unfortunately, it is not helpful to print the nexus_tree_struct since the program works basically with the topology_struct. On the other hand it is easy to change this function to make it work with topology_struct.

Parameters

`in,out`	*fout*	pointer to file handler where tree is to be printed;
`in`	*label*	graph name or any other label;
`in`	*tree*	topology_struct to be printed;

4.29.2.6 apply_spr_at_nodes()

```
void apply_spr_at_nodes (
            topology p,
            topol_node prune,
            topol_node regraft,
            bool update_done )
```

Apply one subtree prune-and-regraft (SPR branch swapping) operation at specified nodes.

Each node is associated to one edge (the branch immediately above it), thus the location of the regraft node will impose the direction of pruning - the prune edge will always detach away from subtree containing regraft. The actual SPR move needs to handle two cases: **prune node is in the path from regraft node to the root** (prune node is least common ancestor between prune and regraft) and **prune node is not in the path from regraft node to root** (prune and regraft nodes share a distinct common ancestor). When prune node is the root, the first case implies in rerooting. Checking against illegal moves (prune==regraft, prune==regraft->up, etc) should be done previous to this function call. This function will call the corresponding lower-level one based on position of prune node. If you know the direction of pruning (rerooting, e.g.) you can call the other two functions directly.

Parameters

`in,out`	*p*	topology over which to apply move
`in`	*prune*	node to be pruned (detached). Direction determined by regraft
`in`	*regraft*	node above which prune node will be reattached

4.29.2.7 apply_spr_at_nodes_notLCAprune()

```
void apply_spr_at_nodes_notLCAprune (
            topology tree,
            topol_node prune,
            topol_node regraft,
            bool update_done )
```

Apply one SPR branch swapping at specified nodes when subtree to be pruned is below prune node.

prune is not lca: Detach the prune subtree and reinsert it just above the regraft node (regraft node may be root).

```
*  Prune:
*
*  p.left\              /p.up.up                      p.left\               |p.up.up
*         \prune___p.up/                       ==>           \p________prune  |
*         /             \                                    /                |
* p.right/              \p.up.left || p.up.right      p.right/                |p.up.left || p.up.right
*
*
```

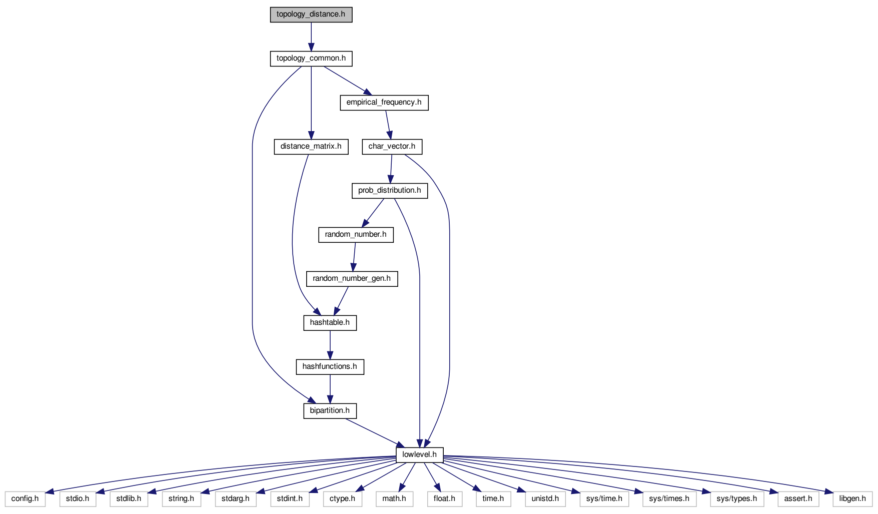
```
*  Regraft:
*
*  p.left\                |r.up        p.left\               /prune.up (=r.up.up)
*         \p________prune |      ==>          \p________prune/
*         /               |                   /              \
* p.right/                |r          p.right/               \r
*
*
```

4.30 topology_distance.h File Reference

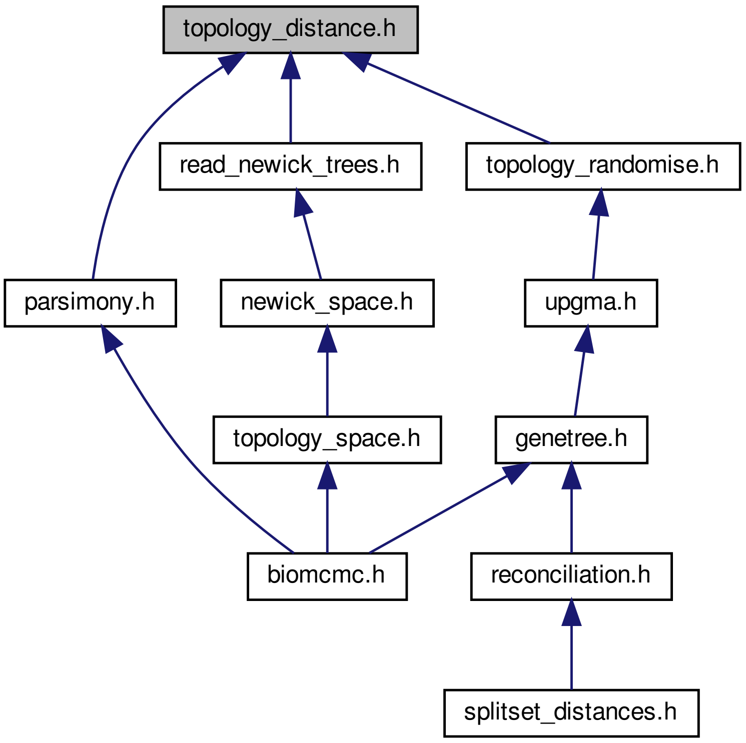
branch length operations on topologies, including patristic distances

```
#include "topology_common.h"
```

Include dependency graph for topology_distance.h:



This graph shows which files directly or indirectly include this file:



Functions

- distance_matrix new_distance_matrix_for_topology (int nleaves)

  allocate memory for a new distance_matrix that will be used on topologies
- void fill_distance_matrix_from_topology (distance_matrix dist, topology tree, double ∗blen, bool use_upper)

  fill in distance_matrix with the patristic distances from topology (can be used with distinct branch length vectors to fill upper and lower diagonals
- void patristic_distances_from_topology_to_vectors (topology tree, double ∗∗dist, double ∗scaling, int n_dists, double tolerance)

  calculates rescaled patristic distances returning up to 6 distinct 1D vectors #dist (externally allocated) The 'tolerance' is the minimum branch length to be considered a multifurcation (length zero)
- int ∗ create_vector_with_idx_leaves_below_for_patristic (topology tree)

  similar to an Euler tour, has list of leaves below each node
- void **estimate_topology_branch_lengths_from_distances** (topology tree, double ∗dist)
- double ∗ **new_topology_branch_lengths_from_distances** (topology tree, double ∗dist)
- void **correct_negative_branch_lengths_from_topology** (topology t, double ∗blength)

4.30.1 Detailed Description

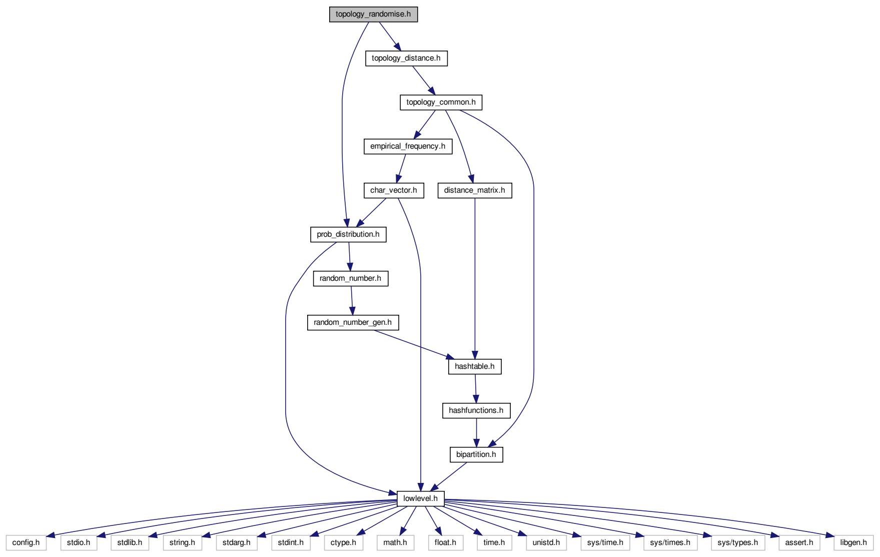
branch length operations on topologies, including patristic distances

4.31 topology_randomise.h File Reference

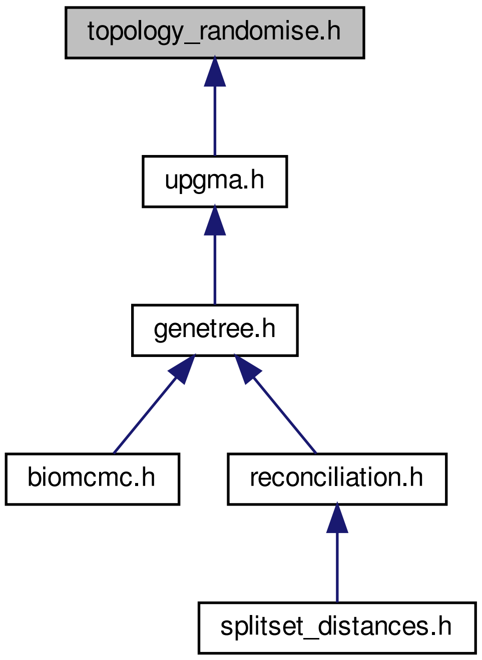
Creation of random topologies and modification of existing ones through branch swapping.

```
#include "topology_distance.h"
#include "prob_distribution.h"
```

Include dependency graph for topology_randomise.h:



This graph shows which files directly or indirectly include this file:



Functions

- void randomise_topology (topology tree)

  low level function that generates a random tree (equiv. to random refinement of a star topology)
- void quasi_randomise_topology (topology tree, int sample_type)

  generates a random topology if sample_type==0, but can reuse some info later to create a "correlated" tree
- void create_parent_node_from_children (topology tree, int parent, int lchild, int rchild)

  create internal node with given children (children coalesce into parent node)
- void topology_apply_rerooting (topology tree, bool update_done)

  random rerooting
- void topology_apply_shortspr (topology tree, bool update_done)

  recursive SPR over all internal nodes, assuming common prob of swap per node
- void topology_apply_shortspr_weighted (topology tree, double ∗prob, bool update_done)

  recursive SPR over all internal nodes, using prob[] vector as rough guide of error rate for node
- void topology_apply_spr_on_subtree (topology tree, topol_node lca, bool update_done)

  random Subtree Prune-and-Regraft branch swapping for subtree below lca node
- void topology_apply_spr (topology tree, bool update_done)

  random Subtree Prune-and-Regraft branch swapping
- void topology_apply_spr_unrooted (topology tree, bool update_done)

  random Subtree Prune-and-Regraft branch swapping generalized (neglecting root)
- void topology_apply_nni (topology tree, bool update_done)

  random Nearest Neighbor Interchange branch swapping (SPR where regraft node is close to prune node)
- bool cant_apply_swap (topology tree)

  check if it is possible to apply SPR/NNI without rerooting (used by topology_apply_spr() and MCMC functions)

4.31.1 Detailed Description

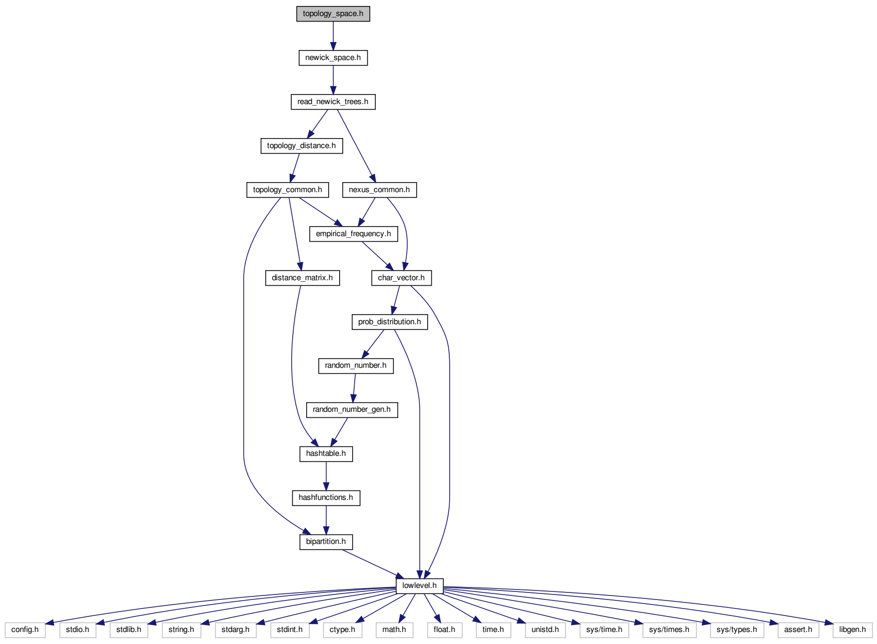
Creation of random topologies and modification of existing ones through branch swapping.

4.32 topology_space.h File Reference

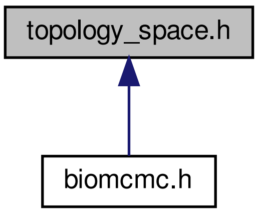
Reads tree files in nexus format and creates a vector of topologies.

```
#include "newick_space.h"
```

Include dependency graph for topology_space.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct topology_space_struct

  Collection of topologies from tree file. When topologies have no branch lengths we store only unique topologies.

Typedefs

- typedef struct topology_space_struct ∗ **topology_space**

Functions

- void add_string_with_size_to_topology_space (topology_space *tsp_address, char *long_string, size_↩
t string_size, bool use_root_location)

  Read tree in newick format until char string_size, returning updated topolgy_space. Auxiliary for python module.
- void add_topology_to_topology_space_if_distinct (topology topol, topology_space tsp, double tree_weight, bool use_root_location)

  Add topology to topology_space only if unrooted version is distinct, updating freqs, trees[] etc. Aux for python module.
- topology_space read_topology_space_from_file (char *seqfilename, hashtable external_taxhash, bool use_root_location)

  Read tree file and store info in topology_space_struct with possible external hashtable to impose the leaf ordering.
- topology_space read_topology_space_from_file_with_burnin_thin (char *seqfilename, hashtable external↩
_taxhash, int burnin, int thin, bool use_root_location)

  lower level function where we can specify burnin and thinning factor, in iterations
- void merge_topology_spaces (topology_space ts1, topology_space ts2, double weight_ts1, bool use_root↩
_location)

  merge trees from two topology_space objects, assuming names hashtable is the same
- void **sort_topology_space_by_frequency** (topology_space tsp, double *external_freqs)
- void save_topology_space_to_trprobs_file (topology_space tsp, char *filename, double credible)

  Save topology_space to a file, in format nexus w/ trprobs, up to "credible" cummul frequency.
- int estimate_treesize_from_file (char *seqfilename)

  Quickly counts the number of leaves in a tree file, without storing any info. Assumes file and trees are well-formed.
- topology_space new_topology_space (void)

  Allocates memory for topology_space_struct (set of trees present in nexus file).
- void del_topology_space (topology_space tsp)

  Free memory from topology_space_struct.

4.32.1 Detailed Description

Reads tree files in nexus format and creates a vector of topologies.

The topology_space_struct is distinct from the newick_space_struct since all trees here must share same char_↩
vector (newick spaces can have general, uncomparable trees), and also since we store the distribution of trees (that is, each tree will have a frequency/representativity associated to it, as typical from Bayesian analyes).

4.32.2 Function Documentation

4.32.2.1 add_string_with_size_to_topology_space()

```
void add_string_with_size_to_topology_space (
            topology_space * tsp,
            char * long_string,
            size_t string_size,
            bool use_root_location )
```

Read tree in newick format until char string_size, returning updated topolgy_space. Auxiliary for python module.

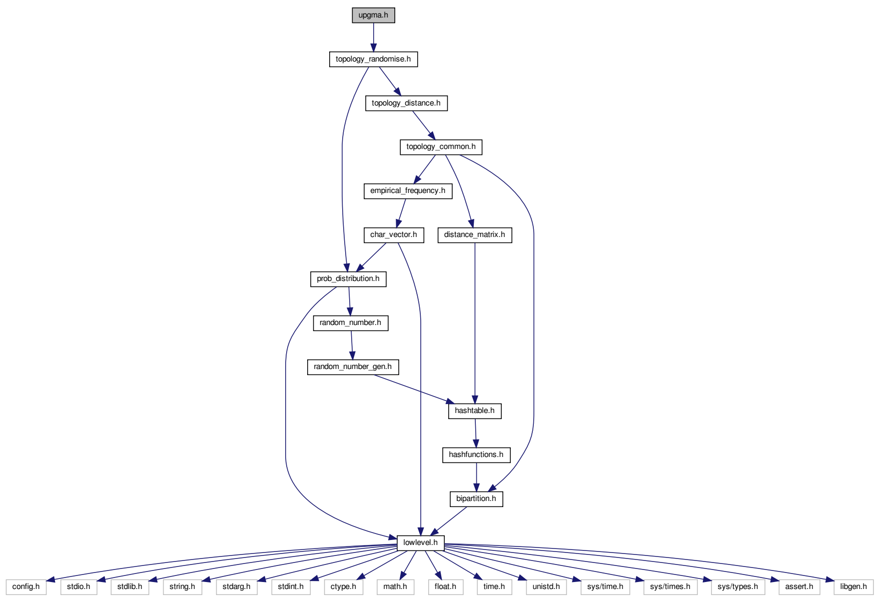
Read tree in newick format until char string_size, returning updated topolgy_space. Auxiliary for python module.

4.33 upgma.h File Reference

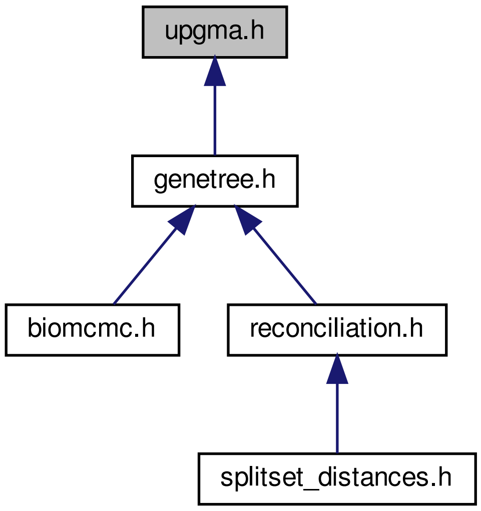
UPGMA and bioNJ from (onedimensional representation of) distance matrices.

`#include "topology_randomise.h"`

Include dependency graph for upgma.h:



This graph shows which files directly or indirectly include this file:



Functions

- void upgma_from_distance_matrix (topology tree, distance_matrix dist, bool single_linkage)

  lowlevel UPGMA (or single-linkage) function that depends on a topology and a matrix_distance
- void bionj_from_distance_matrix (topology tree, distance_matrix dist)

  lowlevel bioNJ function (Gascuel and Cuong implementation) that depends on a topology and a matrix_distance

4.33.1 Detailed Description

UPGMA and bioNJ from (onedimensional representation of) distance matrices.

Index